Universidade de São Paulo
Instituto de Matemática e Estatística
Curso de Bacharelado em Ciência da Computação

Guilherme Camilo Amantéa
Guilherme Grimaldi Nepomuceno

Estudo e Implementação de Redundância dos Serviços na Rede do IME

São Paulo
2007

Guilherme Camilo Amantéa
Guilherme Grimaldi Nepomuceno

Estudo e Implementação de Redundância dos Serviços na Rede do IME

Monografia apresentada ao Programa de Graduação em Bacharelado em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo, como requisito parcial para a obtenção do título de BACHAREL em Bacharelado em Ciência da Computação.

Supervisor: Arnaldo Mandel

Doutor em Mathematics - University of Waterloo, Canada

São Paulo
2007

Resumo

Estuda-se o uso de redundância, principalmente sua aplicação em redes de computadores, como forma de garantir balanceamento de carga e tolerância a falhas. Além disso, descreve-se a implementação de uma de suas formas na rede do Instituto de Matemática e Estatística da Universidade de São Paulo, disponibilizando-se um guia passo a passo e ferramentas para sua administração.

Palavras-chave: Redundância, tolerância a falhas, balanceamento de carga, balanceamanto de processamento, transparência, monitoramento, *Linux Virtual Server*

"It is a mistake to think you can solve any major problem with just potatoes".

Douglas Adams

Sumário

Lista de Figuras

Parte I

Parte Objetiva

1 Introdução

Este capítulo tem o objetivo de apresentar as idéias principais acerca do estudo e implementação de redundância nos serviços da rede do IME.

Nos dias de hoje, cada vez mais, instituições precisam manter seus serviços e sistemas disponíveis. Uma interrupção nessa disponibilidade pode gerar inúmeros transtornos e, em alguns casos, prejuízos. Durante uma cirurgia complicada, a saúde do paciente depende de que todos os aparelhos de suporte à vida funcionem ininterruptamente. Uma empresa que atua na bolsa de valores pode perder milhões se seus sistemas falharem mesmo por alguns minutos. Clientes não hesitariam em trocar de *site* de compras na internet caso este saia do ar.

No caso específico de uma rede de computadores, dizemos que o sistema está disponível quando usuários podem acessá-lo. Caso contrário, esse sistema é dito indisponível. Tal indisponibilidade pode ser:

- imprevista, como quando há queda de energia, defeito em algum componente do sistema (falha na CPU, memória RAM queimada, mau contato, corte em cabos de rede) ou *bug* em algum *software*;
- prevista, que decorre de manutenção que necessita desligar alguns dos componentes, como, por exemplo, atualizações de *software* que requerem reinicialização, *upgrades* de memória ou de disco.

Um sistema tolerante a falhas garante uma alta disponibilidade pois é capaz de continuar funcionando mesmo quando alguns de seus componentes falham por qualquer motivo (planejado ou não). Em uma rede de computadores o objetivo é manter serviços como DNS, E-mail, NFS e web disponíveis durante todo o tempo.

Um dos métodos para garantir essa propriedade é a implementação de redundância dos servidores e, conseqüentemente, dos serviços que estes oferecem. Assim, um serviço é dito redundante quando este está instalado em mais de um servidor.

A rede do IME mantém diversos tipos de serviços, tanto para sistemas UNIX quanto para Windows. Há situações em que alguns desses serviços ficam indisponíveis,

seja por falhas ou por necessidade de manutenção. Isso é muito ruim para os usuários - professores, pesquisadores, alunos de pós-graduação e de iniciação científica - que precisam da rede para realizar suas atividades acadêmicas.

1.1 Objetivo

O objetivo principal é garantir que os serviços estejam disponíveis 100% do tempo. Desligamentos ou falhas de servidores devem ocorrer de forma transparente para o usuário. Para isso, os seguintes itens devem foram considerados:

- Estudo dos métodos mais conhecidos de redundância atualmente;
- Determinação de quais desses métodos se adequam às condições e necessidades da rede do IME;
- Teste de viabilidade em laboratório;
- Implementação na rede do IME.

2 Conceitos

Neste capítulo iremos abordar mais a fundo alguns dos conceitos usados neste trabalho, considerando sua aplicação em componentes e sistemas específicos de computadores ou redes de computadores.

2.1 Tolerância a falhas

Como já foi visto anteriormente, a alta disponibilidade de um sistema, seja ele uma rede computadores ou um único computador, visa garantir seu funcionamento operacional durante o maior tempo possível. Embora seja muito difícil e custoso garantir seu funcionamento durante 100% do tempo, existem maneiras de implementá-lo que nos permitem chegar muito próximo desse número.

Um método muito usado na criação de sistemas altamente disponíveis é conhecido como tolerância a falhas. Seu objetivo é certificar que o dado sistema continuará funcionando, com uma possível queda de performance, caso algumas de suas partes falhem, ao invés de permitir que ele falhe completamente. Além disso, a queda de performance decorrente deverá ser proporcional à gravidade da falha, ao contrário de projetos inocentes em que uma pequena falha é capaz de causar uma interrupção total.

Como exemplo, podemos considerar dois protocolos de comunicação de rede, o TCP (*Transmission Control Protocol*) e o UDP (*U D Protocol*). É uma preocupação do TCP garantir a integridade da mensagem trocada por duas máquinas em um canal de comunicação suscetível a falhas e/ou sobrecarregado como uma rede de computadores, mesmo que isso signifique uma perda de velocidade. O UDP é bem mais leve e rápido, porém ele não só não garante integridade como também há perda de pacotes e a comunicação pode falhar completamente.

Falhas são os motivos que geram a indisponibilidade em um sistema e, portanto, não decorrem apenas de defeitos e problemas imprevistos de seus componentes. Muitas vezes manutenções e atualizações podem gerar indisponibilidade, que caracterizam uma falha.

Para se obter tolerância a falhas, é necessário prever os possíveis defeitos que cada um dos componentes do sistema pode apresentar e, de uma forma geral, buscar uma forma de preveni-los ou, quando isso não for possível, contorná-los quando ocorrerem. Porém, podem ocorrer situações de proporções catastróficas das quais o sistema não pode se recuperar e, neste caso, a solução mais viável é a reversão para um ponto anterior de funcionamento, como por exemplo, a restauração de *backup*. Isso nos leva a estabelecer os seguintes pré-requisitos para sua implementação:

- Nenhum ponto único de falha: não se deve criar um sistema onde a falha em um único componente é capaz de derrubá-lo completamente. Um único servidor que contém todos os serviços de uma rede, ou um único *switch* ao qual todas as máquinas estão ligadas são exemplos de pontos únicos de falha;
- Nenhum ponto único de reparo: assim como no item anterior, o sistema deve ser capaz de continuar funcionando enquanto manutenções, *upgrades* e, no caso de ocorrência de falha, reparos são feitos;
- Isolamento e contenção de falhas: quando uma falha ocorre, o sistema deve ser capaz de impedir que o componente defeituoso seja acessado, garantindo continuidade de funcionamento, e que o problema se espalhe para o resto de seus componentes;
- Modos de recuperação e reversão: deve ser possível restaurar o sistema a um estado sadio após uma falha. Em última instância, deve-se considerar o uso de *backup*.

No entanto, a tolerância a falhas apresenta algumas desvantagens operacionais, nem sempre claras à primeira vista:

- Custo: componentes tolerantes tendem a aumentar o custo da solução como um todo. Esse custo não é necessariamente financeiro, podendo ser, por exemplo no caso de um parque de servidores, espaço para acomodá-los;
- Detecção de falhas: faz-se necessário um agente extra, capaz de detectar a ocorrência de falhas a que ele próprio estará sujeito;
- Redução de prioridade no reparo de falhas: dado que o sistema continua funcionando após uma falha, a importância dada a sua correção provavelmente será diminuída. Se não for corrigida, o sistema corre o risco de falhar completamente;

- Componentes inferiores: sistemas tolerantes a falha permitem o uso de componentes de qualidade inferior ou até mesmo descartáveis, com a vantagem de diminuir o custo. Porém, o uso de muitos desses componentes pode comprometer sua confiabilidade ao ponto de se igualar a um sistema sem tolerância.

Conclui-se, então, que implementar tolerância a falhas em todos os componentes de um sistema não é viável, devido ao custo de implantação e manutenção. Assim os seguintes critérios podem ajudar na decisão de quais componentes devem ser tolerantes a falhas:

- Quão importante é o componente para o sistema?
- Com que freqüência o componente apresenta falhas?
- Quão custoso é torná-lo tolerante a falhas?

2.2 Redundância

A forma mais comum utilizada para obtenção de tolerância a falhas em um sistema é a implementação de redundância.

Redundância é um método que utiliza dois (ou mais) componentes idênticos de um sistema para aumentar sua capacidade de realizar suas operações por um longo período de tempo, ou seja, sua confiabilidade. Assim, se um desses componentes (ou mais) falhar, o sistema continuaria funcionando mesmo que, algumas vezes, com sua capacidade reduzida.

Podemos enxergar um único computador como um sistema, e suas muitas partes como seus componentes e subcomponentes, como CPU, memória, disco rígido, placa de rede etc. Esses componentes são sujeitos a diversos tipos de falhas, que com a devida replicação, podem ser evitadas. Duas fontes de alimentação, por exemplo, evitam que o computador pare quando uma delas queimar. O uso de redundância de disco rígido, ou RAID (*Redundant Array of Independent Drives*), é um método muito eficaz de manter a integridade dos dados e evitar que, no caso de algum problema em um deles, tais dados não sejam perdidos. Um terceiro exemplo de redundância em computadores é o uso de duas ou mais placas de redes que forma uma única placa virtual, comumente chamada

bonding, que permite não só que a maquina continue acessível no caso de problemas em uma delas, como também uma distribuição melhor da carga e aumento do *throughput*.

Se pensarmos agora em uma rede de computadores como o sistema, podemos enxergar os computadores como seus componentes. Neste caso, serviços oferecidos por estes computadores, como DNS (*Domain Name Service*), cadastro de usuários, e-mail, banco de dados ou qualquer aplicação essencial devem ser redundantes. Em um banco, os dados das contas dos clientes não podem ficar em uma única máquina, nem o acesso a ela ser único. Uma página de comércio eletrônico pode perder muito dinheiro nas poucas horas que seu único servidor está em manutenção. Em ambos os exemplos, o uso de redundância de servidores seria a solução. Essa técnica, chamada *cluster*, nada mais é que agrupar os computadores, chamados de servidores reais, de forma que eles sejam vistos como um único computador virtual.

Os *clusters* têm diversos propósitos, de aumento de performance a tolerância a falhas. Se considerarmos este último caso, a alta disponibilidade é garantida com servidores redundantes que continuam provendo o serviço mesmo quando outro componente falha. Desta forma, *clusters* de alta disponibilidade eliminam pontos únicos de falha intrínsecos do sistema.

Normalmente, esses *clusters* são compostos de dois servidores, o número mínimo que provê redundância. Além disso, podem ser de dois tipos diferentes, geralmente com propósitos distintos:

- Duplicação, em que vários servidores redundantes trabalham concomitantemente, provendo instâncias múltiplas do mesmo serviço e direcionando cada requisição para um, utilizando algum tipo de escalonador, ou para todos e escolhendo o resultado mais apropriado.

- Estepe, em que mais de uma instância do servidor real existe no sistema, porém permanecem desligadas ou ociosas. No caso de uma falha, uma dessas instância é acionada e toma o lugar da original. Esse método, porém, possui uma desvantagem clara no fato de que uma máquina fica parada, sem contribuir para a performance da rede.

Fica evidente, no entanto, que tanto na duplicação quanto no estepe, é necessário o uso de agentes externos capazes de controlar seu funcionamento, seja o esca-

lonador que decide qual computador será usado ou até mesmo um operador que liga um estepe quando percebe uma falha em seu servidor principal.

Outro ponto crítico na aplicação de redundância se dá na necessidade de manter os servidores redundantes com exatamente o mesmo conteúdo. Se dois servidores web redundantes, por exemplo, não tiverem o mesmo conteúdo, usuários acessarão páginas diferentes. Servidores duplicados devem, portanto, estar sempre sincronizados e alterações devem ser feitas em todos ao mesmo tempo. Estepes não precisam de atualizações imediatas, porém não podem ser negligenciados. Caso contrário, quando forem efetivamente acionados, seu conteúdo estará defasado.

Por fim, outros componentes de uma rede também devem ser considerados como candidatos a redundância, principalmente cabos, *hubs*, *switches* e roteadores. De nada adiantaria ter computadores redundantes se sua comunicação é interrompida com a queda de um roteador.

2.2.1 Balanceamento de carga

Uma das vantagens da utilização da duplicação de servidores, que decorre diretamente da estrutura, é o balanceamento de carga. Como mais de um servidor provê o mesmo serviço ao mesmo tempo, ambos podem responder requisições independentemente e em paralelo. Todos os pedidos chegam em um ou mais servidores e então são distribuídos aos restantes pelo escalonador. Espalhando-se assim o trabalho entre vários computadores obtém-se um aumento de performance e uma diminuição de tempo computacional significativos se comparado a uma única máquina, com um custo muito menor que de um computador com capacidade semelhante.

O papel do escalonador, então, é decidir, a cada momento, qual servidor real deverá responder o próximo pedido e retransmiti-lo. Alguns exemplos de possíveis métodos são:

- *least-connection*, em que o pedido é enviado ao servidor com o menor número de conexões ativas naquele momento;
- *round-robin*, em que requisições são enviadas aos servidores reais de forma seqüencial e circular;

- *fastest*, em que a capacidade de cada servidor real é medida dinamicamente e o pedido enviado para aquele que tiver o melhor desempenho no momento.

Além disso, servidores reais podem ter pesos associados a eles de forma que recebam uma parte maior ou menor dos pedidos, em relação aos outros. Isso é usado basicamente para tentar igualar servidores com capacidades diferentes.

2.2.2 Transparência

Do ponto de vista do usuário final, todas as alterações nos serviços de uma rede, ou até mesmo em um único computador, que decorrem da aplicação de redundância e de tolerância a falhas, poderiam ser impraticáveis se, a cada uma delas, fosse pedido a ele que usasse um outro componente.

É muito importante que o usuário não perceba que falhas ocorreram ou que máquinas estejam em manutenção. Quando acessa um serviço, falhas devem ocorrer de forma transparente para ele. Mudanças de um servidor defeituoso para um sadio não devem exigir que o usuário tome qualquer ação; ele deve ser redirecionado automaticamente. Essa transparência garante a fluidez e consistência do sistema.

2.3 Monitoração

É fácil ver que a redundância cobre os primeiros dois e principais pré-requisitos da tolerância a falhas em uma rede de computadores, evitando os pontos únicos de falha e de reparo. No entanto, ela sozinha não é capaz de cobrir o isolamento e a contenção das falhas.

Para isso, é necessário introduzir um agente externo confiável capaz de monitorar o andamento da rede e dos servidores que oferecem serviços redundantes. Esses assim chamados monitores devem ser capazes de detectar a ocorrência de uma falha e avisar o responsável, para que este tome a devida providência.

Atualmente existem inúmeros programas cujo papel é exatamente esse: monitorar serviços, reportar falhas e, em alguns casos, tomar ele mesmo providências cabíveis. Talvez o exemplo mais comum em redes de computadores sejam os programas de antivírus, muito usados em distribuições comerciais. Seu papel é monitorar cada componente da

rede, os computadores, e, no caso de uma infecção, isolar o ataque, tentar corrigi-lo e se certificar que ele não vai se alastrar para as outras máquinas.

No caso específico de serviços em uma rede de computadores, os monitores devem ser capazes de verificar se um dado serviço está funcionando e/ou se a máquina em que ele se hospeda não apresenta nenhum problema. Se essa máquina participa de um *cluster* de alta disponibilidade, o monitor deve ser capaz de removê-la temporariamente quando algum erro na continuidade do serviço for detectado e inseri-la novamente quando corrigido.

Para um monitor, não existe distinção entre falhas, erros e manutenção, embora o motivo possa desencadear ações distintas. Quando um servidor perde sua conexão porque sua placa de rede queimou, um monitor pode acionar um administrador para que verifique porque ele não responde. Quando uma manutenção preventiva ou atualização de software é agendada, o monitor é notificado e se certifica apenas de que a máquina não será acessada durante aquele período. Em ambos os casos, sua função é certificar-se de que o sistema esteja sempre funcionando, mesmo que com sua performance reduzida.

3 Solução Adotada

Este capítulo tem o objetivo de ilustrar a solução adotada, sua estrutura e as ferramentas utilizadas para sua implementação.

3.1 Estrutura

O modelo de redundância adotado incorpora duplicação e estepe, e é composto de 4 elementos básicos: um *cluster* de servidores, um *load-balancer* com LVS, ferramentas de monitoramento e balanceamento dinâmico de processamento.

A seguir, explicaremos cada um desses elementos e como eles se encaixam no modelo adotado.

3.2 *Cluster* de servidores

Tendo como intuito a alta disponibilidade, este modelo faz uso de, no mínimo, dois servidores redundantes, que ofereçam os mesmos serviços. Esses servidores são chamados de servidores reais.

Como visto anteriormente, o conteúdo desses servidores deve ser exatamente igual e, para isso, devem estar sempre sincronizados, de forma que atualizações em um deles sejam replicadas aos demais no menor espaço de tempo possível. Como a sincronização manual é trabalhosa e sujeita a erros e atrasos, uma opção viável para a sincronização automatizada é o uso de `rsync`. A performance da sincronização pode ainda ser melhorada com o uso de uma rede dedicada para esse propósito, não sobrecarregando a rede oficial com pacotes de sincronização.

Não há necessidade, no entanto, que tais servidores tenham hardware (CPU, memória etc) iguais ou até mesmo semelhantes. Como veremos mais a frente, o uso de pesos e balanceamento dinâmico de processamento visam equilibrar o uso de máquinas muito diferentes.

Vale lembrar que esses servidores devem ser máquinas independentes e não servidores virtuais em uma mesma máquina. Enquanto é possível que cada uma delas tenha servidores virtuais instalados, o uso desse *cluster* "virtual" destrói o propósito de alta disponibilidade, uma vez que algum problema com a máquina hospedeira removerá todos os servidores redundantes.

3.3 *Linux Virtual Server*

O *Linux Virtual Server[2]* (LVS) é um servidor virtual altamente disponível e escalável construido sobre um *cluster* de servidores reais, com um *load balancer*, chamado diretor, rodando sistema operacional Linux. Ele permite o balanceamento de carga de serviços de rede tais como web, mail e servidores de aplicação.

O LVS é um módulo de *kernel* que implementa chaveamento na camada quatro de rede, a camada de transporte. Isso permite que sessões TCP ou UDP sejam distribuídas de forma inteligente entre uma série de servidores reais. Seu uso mais comum se dá no tráfego HTTP e HTTPS, mas pode ser usado para praticamente todos os serviços, desde servidores de e-mail até servidores de aplicação.

Embora o LVS rode em uma máquina com sistema operacional Linux, ele é capaz de controlar conexões de usuários finais para servidores reais rodando qualquer sistema operacional, contanto que a conexão seja TCP ou UDP.

A arquitetura do *cluster* de servidores é totalmente transparente para o usuário final, que interage com ele como se este fosse um único servidor virtual de alta performance e disponibilidade.

3.3.1 Terminologia

- Diretor: máquina com Linux e LVS instalado que recebe os pacotes dos usuários finais e os encaminha para os servidores reais.
- Usuário final: máquina que origina uma conexão.
- Servidores reais: máquinas na ponta final da conexão, que oferecem algum serviço redundante.

Um único computador pode ter mais de um dos papéis acima ao mesmo tempo.

- Virtual IP (VIP): endereço IP designado para o serviço de que o diretor irá tratar.
- Director IP (DIP): endereço IP do diretor.
- Client IP (CIP): endereço IP de um usuário final.
- Real IP (RIP): endereço IP dos servidores reais.

3.3.2 Funcionamento

Internamente, o LVS possui uma tabela de serviços que são oferecidos pelo servidor virtual. Esses serviços virtuais são definidos por um endereço IP, que os usuários finais usarão para acessá-lo, uma porta, usada na conexão, um protocolo, que pode ser tanto TCP como UDP. Para cada um desses serviços, mantém um escalonador e uma tabela de endereços IP dos servidores reais que os oferecem.

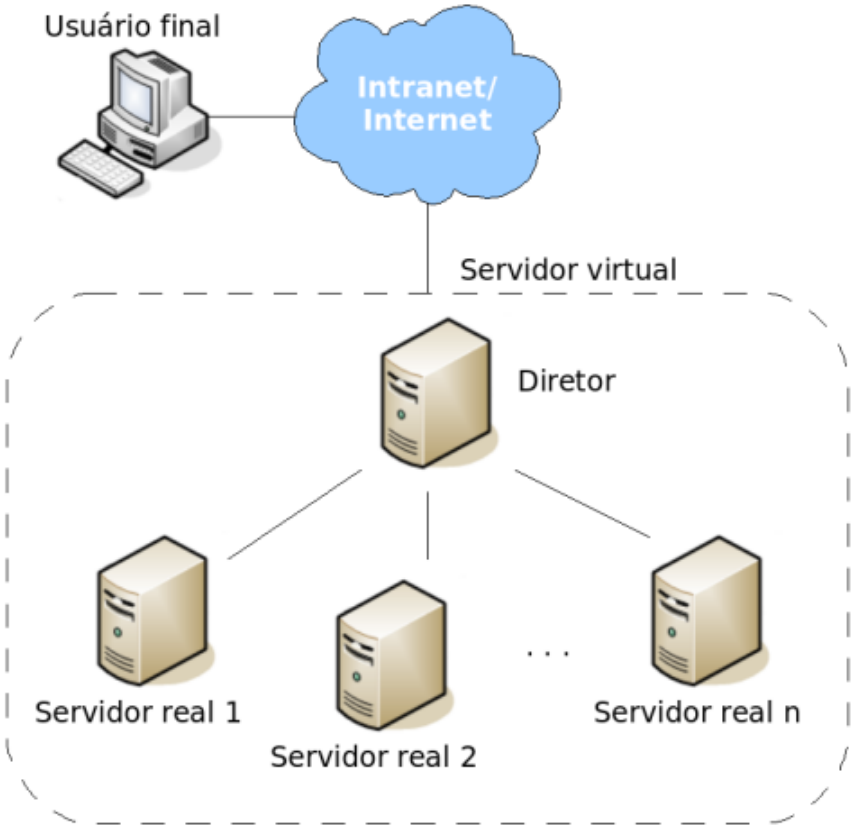

Figura 3.1: Exemplo de estrutura do LVS

Um cliente ou usuário final faz uma requisição para o servidor virtual. Neste momento, o diretor recebe o pacote e consulta sua tabela LVS, verificando quais servidores reais são responsáveis pelo pedido. Baseado no escalonador configurado para esse serviço, o diretor encaminha esse pedido usando um dos métodos de redirecionamento, que veremos mais abaixo. A partir desse momento, os pacotes suseqüentes da mesma conexão serão encaminhados para o mesmo servidor real, certificando-se, assim, que sua integridade será mantida.

Um serviço virtual pode ter, associado a ele, um *timeout* de persistência. Se isso está configurado e uma segunda requisição é recebida a partir do mesmo endereço IP antes de o *timeout* expirar, então esta é encaminhada para o mesmo servidor real da conexão original.

3.3.3 Redirecionamento de pacotes

O LVS possui três formas diferentes de redirecionar os pacotes recebidos:

- *Network Address Translation* (NAT): método de manipulação do endereço e/ou porta de origem e destino de um pacote. Seu uso mais comum é mascarar o IP de redes internas para o acesso a internet. No contexto de chaveamento na camada quatro, pacotes são recebidos dos usuários finais e a porta de destino e endereço IP são mudados para aqueles do servidor real escolhido. Pacotes retornados passam pelo diretor, que remove a alteração para que o usuário final veja a respota do fonte esperada.

- *Direct Routing* (DR): todas as máquinas encontram-se na mesma rede e os pacotes dos usuários finais são encaminhados diretamente para o servidor real. O pacote IP não é alterado e, então, o servidor real deve ser configurado para aceitar tráfego para o IP do servidor virtual. Isso pode ser feito usando uma interface *dummy* ou um fitro de pacotes para redirecionar o tráfego destinado ao IP do servidor virtual para uma porta local. Os servidores reais podem enviar respostas diretamente de volta para o usuário final, sem passar pelo diretor.

- IP *Tunnelling*: Permite que pacotes destinados a um endereço IP sejam redirecionados para outro endereço, até mesmo em outra rede. No contexto de chaveamento da camada quatro, seu funcionamento é similar ao *direct routing*, porém os pacotes

são encapsulados em um novo pacote IP, ao invés de apenas manipular o quadro *ethernet*. Sua maior vantagem é permitir que os servidores reais estejam localizados em redes diferentes.

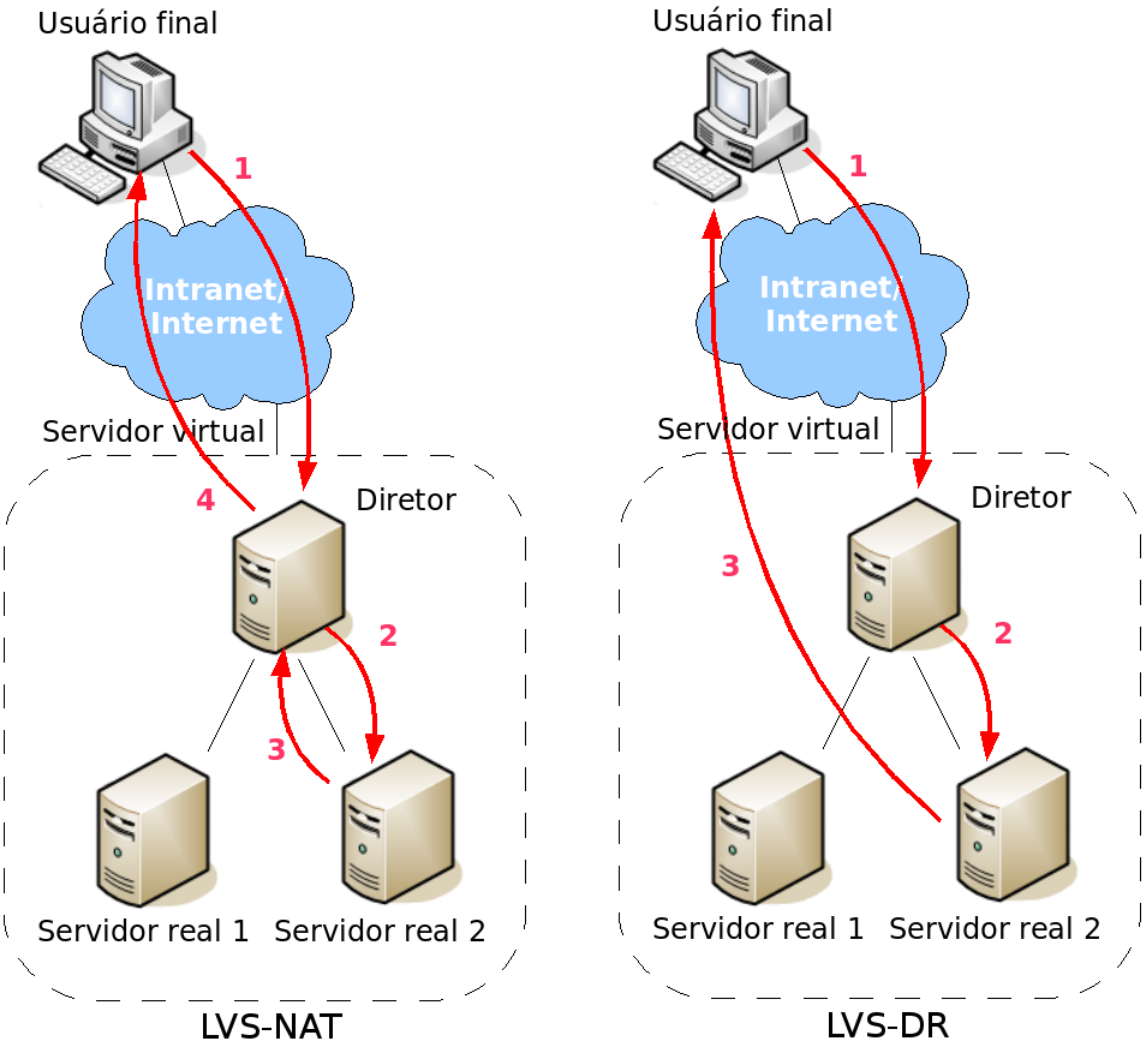

Figura 3.2: Ciclo de conexão através do LVS

3.3.4 Escalonamento

A cada serviço virtual é designado um algoritmo de escalonamento, chamado escalonador, que é usado para alocar requisições entrantes para os servidores reais. No LVS, os escalonadores são implementados como módulos de *kernel* separados, possibilitando a criação, remoção e edição de algoritmos sem alterar o código base do LVS.

O LVS aceita vários escalonadores, como o *round robin* e o *least connection* e também suas versões com pesos. Além disso, escalonadores mais complexos, implementa-

dos para propósitos específicos, podem ser utilizados. Por exemplo, no uso do LVS para balanceamento de carga de *proxies* transparentes, para garantir que requisições para o mesmo endereço IP sejam enviadas para o mesmo servidor real.

3.3.5 Performance

Alguém poderia se perguntar se, pelo fato de cada requisição feita ao servidor virtual criado pelo LVS precisar passar pelo diretor, o fluxo de dados na rede não ficaria muito menos eficiente. Esta sessão tem o objetivo de analisar a performance de desempenho do LVS para responder a essa pergunta.

Analisaremos projetos baseados em LVS-NAT e LVS-DR, pois o LVS-Tunnelling tem um comportamento muito semelhante ao LVS-DR. Essa análise foi baseada no artigo de Patrick O'Rourke e Mike Keefe [3], de onde os gráficos foram tirados. Nessas figuras, os termos em inglês se referem a:

- *Reply Rate*: Número de respostas por segundo;
- *Connections/sec*: Número de conexões por segundo;
- *Direct*: Conexão direta entre cliente e servidor real;
- *n RS*: Número de servidores reais igual a n;
- *Throughput*: Throughput.

Como já se foi mencionado anteriormente, o LVS-NAT tem a desvantagem contra o LVS-DR de que as respostas dos servidores reais às requisições feitas ao servidor virtual devem passar pelo diretor. Como a figura 3.3 mostra, isso implica que apenas um servidor real é suficiente para saturar o diretor, de forma que o aumento do número de servidores reais não melhora muito a performance do servidor virtual.

Porém, em algumas situações, o LVS-NAT pode ser a única maneira disponível de se implementar o LVS. Ela é interessante, pois continua garantindo a alta disponibilidade do servidor virtual. Além disso, apesar da performance mais baixa que a conexão direta, no artigo [3], O'Rourke e Keefe mencionam que, quando é comparado com um balanceador de carga em *hardware* do mercado, o LVS-NAT ainda pode ter um custo, em dinheiro, por requisições por segundo duas vezes menor, o que torna o custo-benefício do LVS-NAT muito bom.

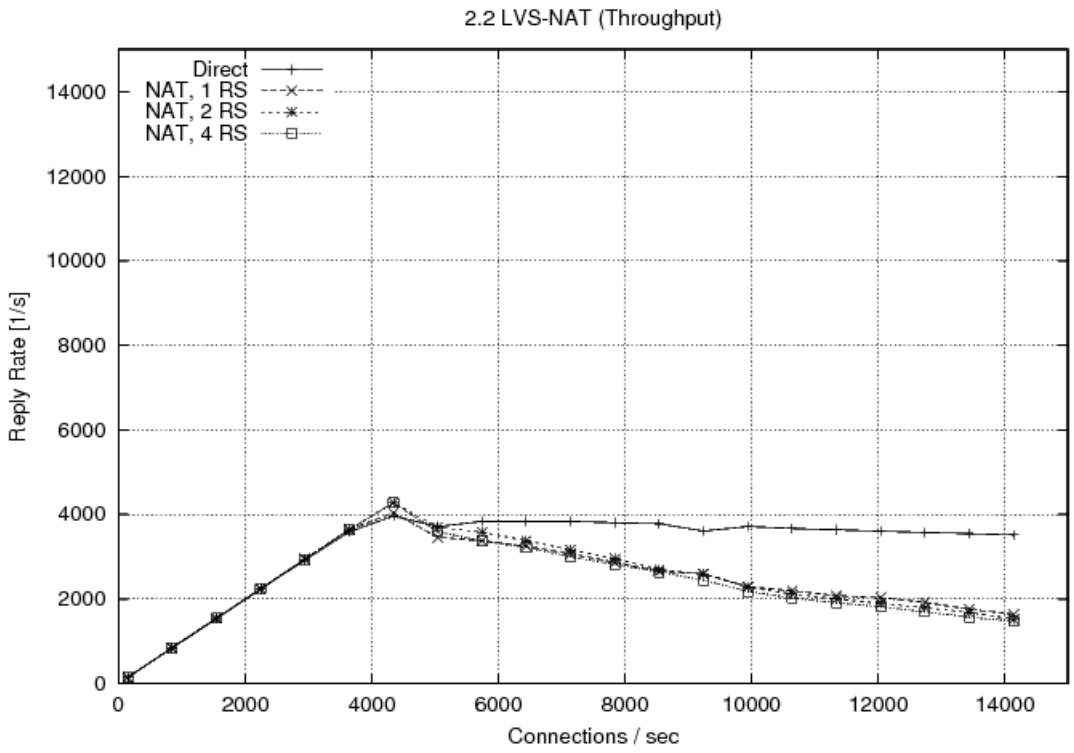

Figura 3.3: LVS-NAT com diretor com 1 processador e *kernel* 2.2

A figura 3.4 mostra o desempenho do LVS-DR implementado em um diretor com um único processador, rodando *kernel* 2.4.

Como podemos perceber, com apenas um servidor real, o desempenho do LVS-DR se assemelha ao daquele com conexão direta entre o cliente e o servidor. Porém, quando se adicionam mais servidores reais, o aumento da performance é considerável. Essa melhora se deve ao balanceamento de carga.

Nas figuras 3.5 e 3.6, vemos que, se o diretor dispuser de mais de um processador, a melhora de performance, ao se adicionar mais servidores reais, se aproxima de ser linear crescente. Isso é útil quando se quer prever em quantos servidores reais deve-se investir quando se pretende aumentar o número de requisições por segundo.

LVS contra DNS

Balanceamento de carga implementado com DNS é, provavelmente, o método mais simples utilizado para se construir um *cluster* de serviços de rede. Ele usa o DNS para distribuir requisições entre diferentes servidores, associando a cada uma um endereço IP possivelmente diferente, segundo alguma estratégia de escalonamento, geralmente simples, como

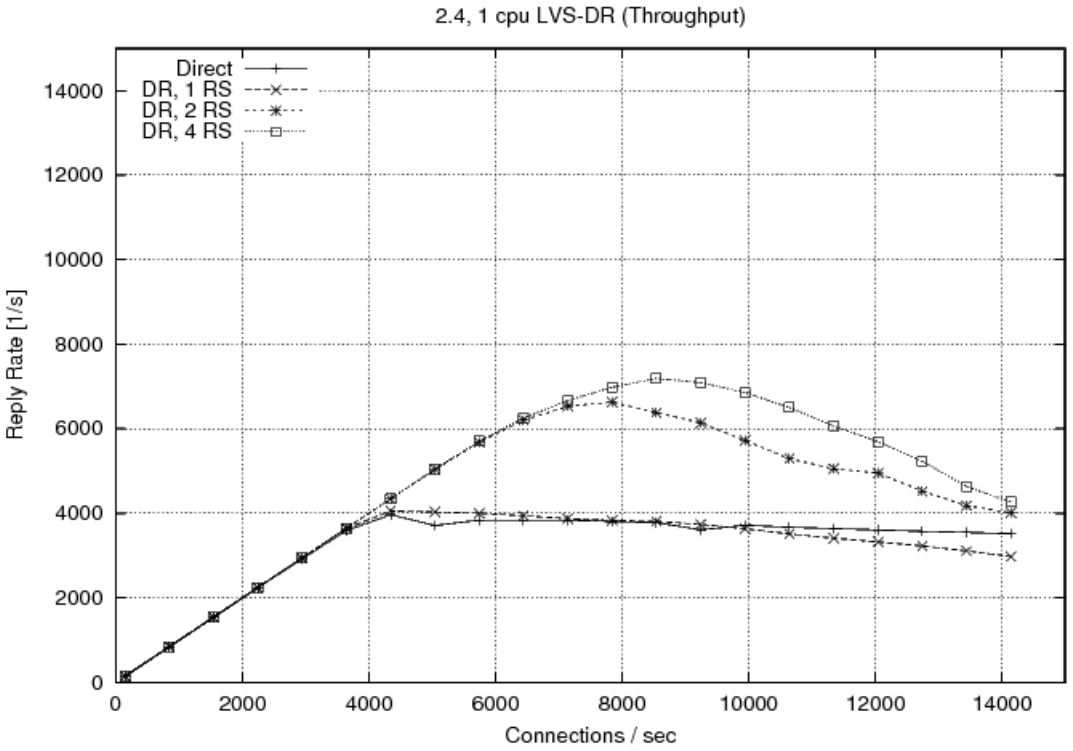

Figura 3.4: LVS-DR com diretor com 1 processador e *kernel* 2.4

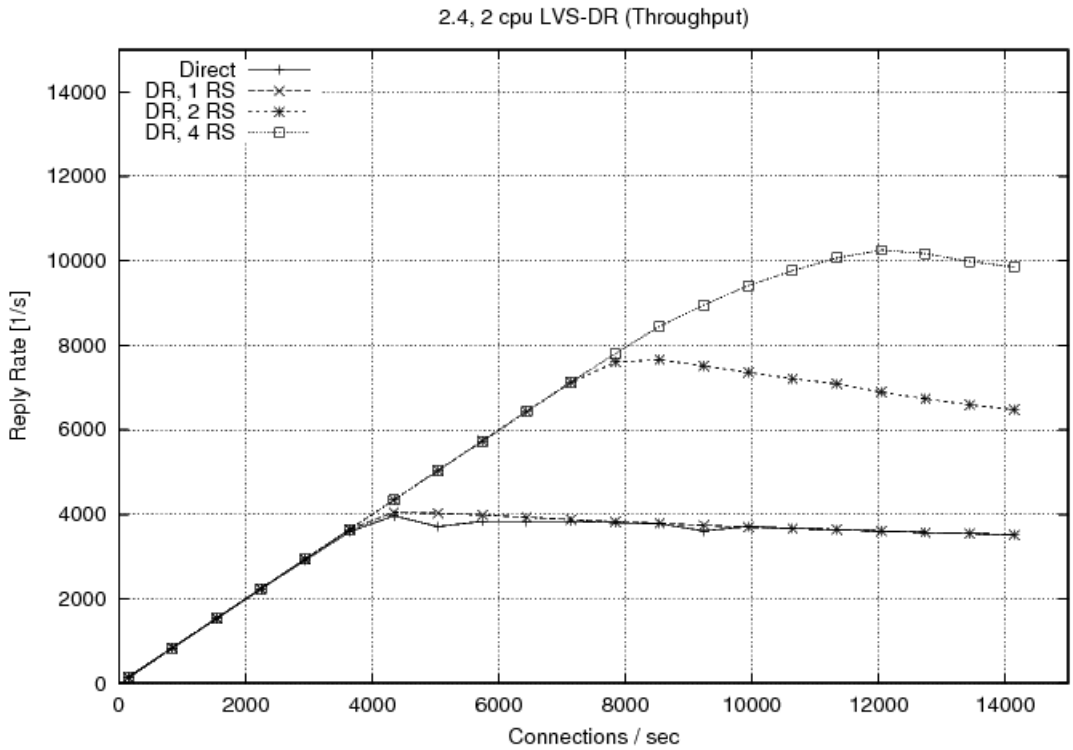

Figura 3.5: LVS-DR com diretor com 2 processadores e *kernel* 2.4

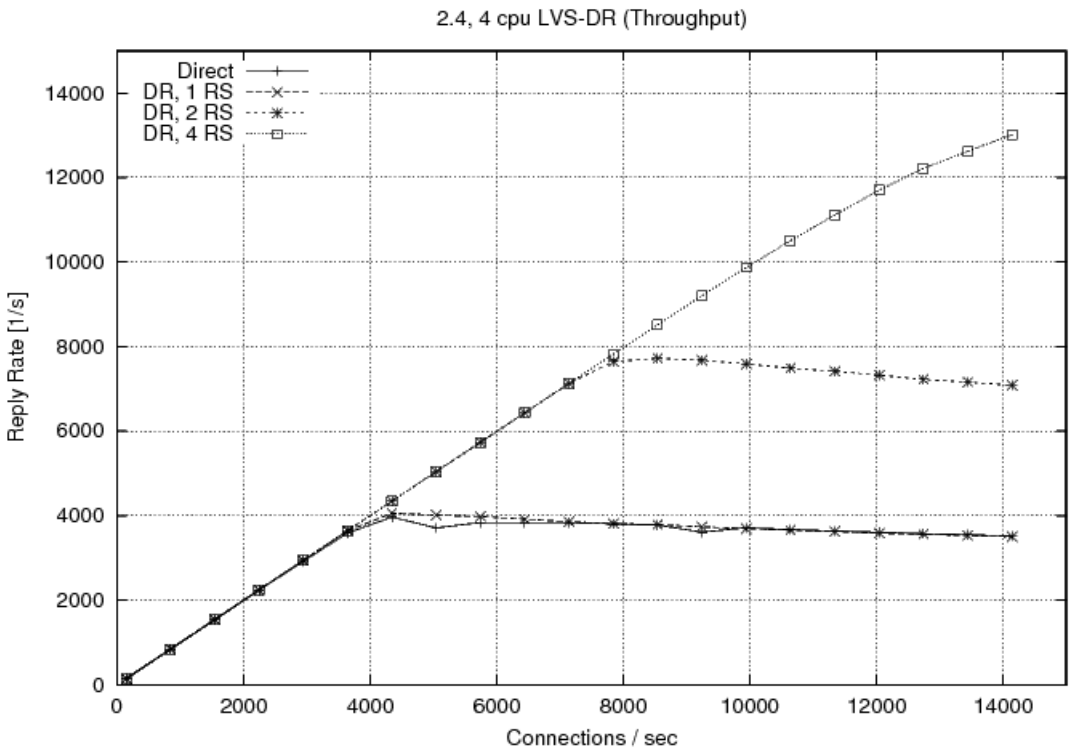

Figura 3.6: LVS-DR com diretor com 4 processadores e *kernel* 2.4

o *round-robin*. Porém, requisições subseqüentes são enviadas para um mesmo servidor dentro de um tempo pré-estabelecido (TTL, de *Time-to-live*).

Não é possível, de antemão, especificar-se o TTL de forma perfeita. Se o configurarmos pequeno demais, o servidor DNS ficará sobrecarregado, pois muitas consultas serão feitas a ele e, por se tratar de de um aplicativo e não um módulo de *kernel*, tem uma performance dramaticamente reduzida, se comparado ao LVS. Se o configurarmos grande demais, haverá um desbalanceamento de carga, pois, em um dado momento, muitas requisições estarão sendo direcionadas para um servidor, enquanto outros estarão ociosos.

Além disso, o LVS já disponibiliza vários algoritmos de escalonamento prontos, como *round-robin*, *least-connection*, suas versões com pesos, outros mais complexos e a possibilidade de implementação específica para uma dada situação.

Podemos concluir então, que o LVS é uma solução robusta e completa, cuja performance é melhor que as soluções mais simples e até mesmo modelos em *hardware*.

3.4 Monitoramento

O que aconteceria se um dos servidores reais parasse de funcionar? Ou pior, o que aconteceria se o próprio diretor falhasse? No primeiro caso, os pacotes dos clientes seriam mandados para um servidor que não responde, e dependendo do escalonador escolhido e das condições da rede, o número desses pacotes pode ser muito grande. Já no segundo caso, todo servidor virtual estaria fora do ar, mesmo com os servidores reais ainda respondendo.

Em ambos os casos acima, a alta disponibilidade estaria comprometida. Como já visto no capítulo anterior, uma parte essencial da implantação de alta disponibilidade é a utilização de algum recurso de monitoramento capaz de detectar a ocorrência de falhas e avisar os módulos responsáveis ou tomar ele mesmo as providências necessárias.

Conclui-se, então, que existem duas necessidades distintas de monitoramento: monitorar os servidores reais, com o intuito de se certificar que todos estejam funcionando e acessíveis, e monitorar o diretor, para que o sistema todo não falhe completamente.

3.4.1 Mon

O monitoramento dos servidores reais é feito com a ferramenta `mon`. Essa ferramenta é capaz de monitorar a disponibilidade de serviços e enviar alertas na ocorrência de eventos especificados.

O `mon` é composto por dois grupos de subprogramas, monitores e alertas. Cada serviço é monitorado por um monitor, que pode ser desde um simples `ping` até uma análise complexa da resposta de uma transação. Alertas são ações como o envio de e-mail ou a ativação de um *backup*, que podem ocorrer tanto no momento da falha quanto quando se restabelece o funcionamento. Essa estrutura é bastante flexível e extensível e permite a criação de monitores e alertas específicos para uma dada situação.

No caso do LVS, é importante garantir que sua tabela esteja sempre correta e atualizada. Servidores reais que sofrem alguma falha devem ser removidos da tabela para que usuários finais não tentem acessá-lo, sem sucesso. Uma vez que esse servidor voltar ao normal, deve ser incluindo novamente na tabela.

Assim, o `mon` deve ser instalado no diretor e, para cada serviço virtual na tabela LVS, monitorar cada um dos servidores reais que o provê. Se o protocolo desse serviço for

TCP, o monitor responsável contata a porta especificada e verifica se há conexão. Se o protocolo for UDP, como não há como verificar a conexão, o teste se resume a um simples `ping` para ver se o servidor está ativo. Em ambos os casos, se o monitor verificar alguma falha, ele aciona um alerta, que é responsável por atualizar a tabela do LVS, removendo o servidor inacessível, gravar um arquivo de log e enviar e-mails para os responsáveis. Quando o erro é corrigido, o monitor verifica isso e novamente aciona o alerta que, desta vez, inclui o servidor na tabela.

Desta forma garantimos que a tabela está sempre atualizada e correta.

3.4.2 Heartbeat

Um dos pré-requisitos principais da tolerância a falha é a não existência de um ponto único de falha. No entanto, no LVS, uma falha no diretor derrubaria todo o sistema. Para evitar que isso ocorra o diretor precisa de redundância.

Essa redundância se dá na forma de um estepe, ou seja, um outro servidor idêntico ao diretor que seja capaz de assumir seu papel quando este não está respondendo. Se a troca de contexto dos dois servidores fosse manual, com um operador verificando o erro e, só então, trocando os papéis, teríamos um período muito grande de *downtime*, o que não é desejado. É, portanto, necessário automatizar esse processo.

O `heartbeat` é uma ferramenta de monitoração mútua. Na sua forma mais básica, é instalado em duas máquinas e roda *scripts* definidos pelo usuário na inicialização e quando uma dessas máquinas cai ou volta a funcionar. Ele funciona trocando `pings` entre as máquinas para verificar se a outra está funcionando e é até mesmo capaz de assumir o endereço IP uma da outra, em caso de falha.

Assim, tanto diretor como estepe possuem o `heartbeat` instalado e rodando. Quando o estepe verifica que o diretor não está respondendo, assume suas funcionalidades, assim como o VIP (IP virtual), e passa não só a receber requisições e encaminhá-las aos respectivos servidores reais como também a monitorá-los através do `mon`. Quando o diretor volta a seu estado normal, recupera suas funcionalidades, o VIP e o monitoramento dos servidores reais.

Como a monitoração do `heartbeat` é mútua, mesmo que ocorra algum problema na execução de alguns dos lados, ou até mesmo ambos, não há risco das duas

máquinas assumirem o mesmo VIP: se o `hearbeat` do diretor falhar, nada acontece pois ele ainda está servindo seu papel; se falhar no estepe, esse simplesmente não perceberá se o diretor está funcionando ou não e, como é o próprio `heartbeat` que assume o VIP, nada acontecerá. Na pior das hipóteses, poderá acontecer do diretor cair, e o estepe não assumir seu lugar, que não deixa de ser indesejável.

Fica claro, então, que há uma grande necessidade manter o `heartbeat` funcionando corretamente. Além disso, vale lembrar que estepe e diretor devem estar sempre sincronizados quanto a configurações, principalmente do `mon`, uma vez que este é capaz de preencher a tabela LVS corretamente quando o estepe assume o diretor, e vice-versa.

3.5 Balanceamento dinâmico de processamento

Geralmente, os servidores reais são máquinas distintas, com hardware, como CPU e memória, nem sempre igual. Fazendo parte de um mesmo *cluster*, problemas podem surgir quando usuários finais acessam máquinas com capacidades diferentes que oferecem o mesmo serviço.

Neste caso, o uso de pesos é essencial. Máquinas mais potentes, que conseguem tratar um número maior de conexões, têm um peso associado a ela maior que as outras. Assim, o número de requisições enviadas a ela é maior e o sistema fica mais balanceado. O LVS leva isso em consideração quando usa escalonadores como *weighted round robin* ou *weighted least connection*, registrando em sua tabela o peso de cada servidor real para cada serviço.

No entanto, pesos não são capazes de levar em conta a carga atual de uma determinada máquina. Se a melhor máquina, que tem o maior peso, estiver sobrecarregada, mesmo assim, a maior parte das requisições será enviada a ela, sobrecarregando-a ainda mais, enquanto as outras podem estar até mesmo ociosas. É desejável, então, garantir que usuários finais utilizem sempre as máquinas com recursos menos ocupados.

O balanceamento dinâmico de processamento consiste em verificar a utilização de cada um dos servidores reais freqüentemente e atualizar seus pesos na tabela do LVS.

Assim nenhuma máquina é sobrecarregada com poucos clientes que têm alta demanda, e o sistema fica equilibrado.

4 Implementação

Neste capítulo descreveremos como a solução adotada foi implementada na rede do IME.

4.1 A rede do IME

A rede do IME mantém diversos tipos de serviços, tanto para sistemas UNIX quanto para Windows. Alguns deles já possuem algum tipo de redundância, como o NFS e servidores de aplicação.

Além disso, há situações em que alguns serviços ficam indisponíveis. Isso é muito ruim para os muitos usuários - professores, pesquisadores, alunos de pós-graduação e de iniciação científica - que precisam da rede para realizar suas atividades acadêmicas.

Um dos principais motivos para a aplicação de redundância na rede do IME é o desbalanceamento de carga entre os servidores de aplicação (gerenciadores de área de trabalho). Embora já haja duplicação dos servidores (atualmente existem quatro deles), não há nenhum critério automático de escalonamento. Os usuários escolhem qual servidor usar através de uma lista a ele apresentada. Na prática, isso tem gerado maior concentração de uso em apenas um desses servidores, seja por preferência, desconhecimento ou por selecionar o primeiro da lista, quando está com pressa.

4.2 Estrutura física

Como vimos no capítulo anterior, a solução adotada faz uso do LVS, `mon`, `heartbeat` e balanceamento dinâmico de processamento. Como a rede do IME está estruturada de forma que todas as máquinas pertencem à mesma rede física, o LVS-DR é o esquema de redirecionamento mais indicado.

Apesar do intuito inicial ser a utilização do LVS em servidores de aplicação, a implementação da solução é genérica e permite o uso de praticamente qualquer serviço TCP ou UDP. Ela usa como endereço do servidor virtual o VIP 192.168.45.26, e apresenta-se na figura 4.1.

- **Diretor**: instalado na máquina de nome `midas`, com DIP 192.168.45.23;
- **Servidores Reais**: máquinas de nome `kama`, com RIP 192.168.45.43, e `hades`, com RIP 192.168.45.13;
- **Estepe**: instalado na máquina de nome `estepe`, com DIP 192.168.45.204.

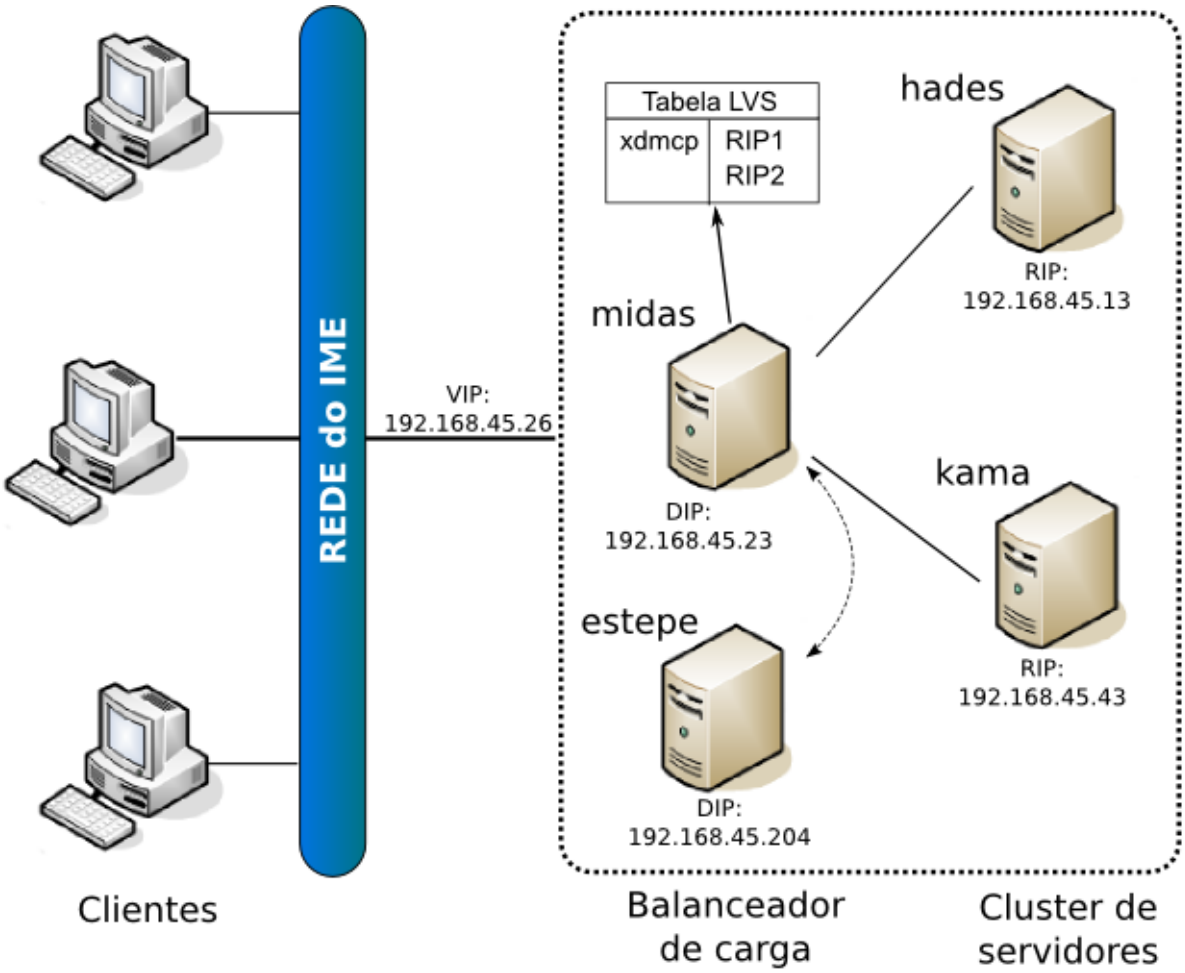

Figura 4.1: Implementação do LVS-DR na rede do IME

Essa implementação foi testada em ambiente controlado com apenas um cliente e mais dois servidores reais de `ssh`, aplicação e web. Atualmente, encontra-se em fase de teste de produção, ou seja, uma parcela dos usuários da rede estão utilizando-a e avaliando-a.

O serviço de aplicação `xdmcp` trabalha com o protocolo UDP e TCP. Inicialmente, o cliente (servidor X) faz uma requisição UDP para o servidor de aplicação no LVS; essa requisição passa pelo diretor, que garante a redundância e o balanceamento. Uma vez autenticado, o servidor de aplicação inicia uma conexão TCP com o cliente. A partir desse momento, o diretor não participa mais do processo.

O balanceamento dinâmico de processamento implementado utiliza, nos servidores reais, o *script* `uptimesrv.pl` para criar um processo em plano de fundo que ouve em uma porta específica (no caso, a porta 2357) e responde a requisições com o resultado de `uptime`, que diz a porcentagem de uso da máquina. Essa solução está intrinsecamente ligada ao sistema operacional Linux. No diretor, um *script* freqüentemente acessa cada um dos servidores reais na porta acima e utiliza as informações obtidas para atualizar os pesos na tabela LVS.

4.2.1 O problema ARP

Normalmente, quando uma máquina inicia uma conexão TCP/IP, ela manda um pacote ARP (por *broadcast*) perguntando quem possui o endereço IP requisitado. A máquina destino responde, então, com seu endereço MAC e a conexão é estabelecida. A configuração do LVS-DR faz com que o diretor e os servidores reais tenham o mesmo VIP e, portanto, uma requisição ARP pode receber, como resposta, endereços MAC de todos eles.

Assim, um cliente pode se conectar diretamente a um servidor real, ignorando o LVS. Ainda pior é o fato de que, durante uma seção TCP, ele pode receber o endereço de outro servidor real como resposta a uma nova requisição ARP (devido a seu *timeout*) e se conectar a um servidor do qual não tem nenhuma informação.

Para resolver o problema é necessário garantir que apenas o diretor preste atenção em requisições ARP direcionadas para o VIP (o DIP e os RIPs devem funcionar normalmente). Desta forma, ele será o único a respondê-las.

4.2.2 `ipvsadm`

A ferramenta mais básica de manipulação do LVS é o `ipvsadm`. Ela manipula as tabelas LVS diretamente e permite adicionar ou remover serviços virtuais, adicionar, remover e alterar entradas de servidores reais e seus pesos (quando pertinente) e visualizar estatísticas de uso. Os *scripts* mostrados na próxima seção utilizam o `ipvsadm` para configurar o LVS no diretor.

4.3 Estrutura do diretório `lvs`

A seguir mostraremos os arquivos criados para a manutenção e suporte do LVS, com o intuito de facilitar a administração. Os *scripts* foram desenvolvidos em `perl` e `bash`.

- `lvs/`

  Diretório raiz.

- `lvs/bin`

  Diretório de arquivos executáveis.

- `lvs/bin/add_service`

  Adiciona um serviço a tabela do LVS.

- `lvs/bin/fill_table`

  Preenche a tabela do LVS, de acordo com as configurações especificadas pelo administrador no arquivo `lvs.conf`.

- `lvs/bin/director.lvs.dr`

  Configura as interfaces de rede de uma máquina e executa `fill_table` para torná-la um diretor em um esquema LVS-DR.

- `lvs/bin/configure.pl`

  Lê o arquivo de configuração `lvs.conf`, escreve `fill_table` e escreve o arquivo de configuração do `mon`, preparando, assim, o diretor.

- `lvs/bin/lvs.alert`

  Alerta usado pelo `mon` que remove as entradas de um servidor real, se ele falhou, e adiciona-as quando este voltar. Além disso escreve um arquivo de log com a data, o nome do servidor e o serviço.

- `lvs/bin/up.logger`

  Registra em um arquivo de log, a hora, data, serviço e nome do servidor real que voltou ao ar.

- `lvs/bin/down.logger`

  Registra em um arquivo de log, a hora, data, serviço e nome do servidor real que falhou.

- `lvs/etc`

  Diretório de arquivos de configuração.

- `lvs/etc/lvs.conf`

  Contém todas as informações sobre os serviços e servidores reais, necessárias para o LVS e para o `mon`.

- `lvs/etc/ha.d`

  Diretório que contém exemplos das configurações do `heartbeat`, que devem ser iguais tanto no diretor quanto no estepe.

- `lvs/util`

  Diretório de utilitários.

- `lvs/util/realserver.lvs.dr`

  Configura as interfaces de rede de uma máquina para torná-la um servidor real em um esquema LVS-DR. Esse *script* deve ser executado nos servidores reais.

- `lvs/util/poll.pl`

  Coleta informações sobre o uso de recursos dos servidores reais e atualiza os seus pesos na tabela LVS.

- `lvs/util/uptimesrv.pl`

  Servidor de uptime que roda nos servidores reais.

- `lvs/var`

  Diretório de variáveis para IPC.

- `lvs/var/polling`

  Contém os pesos dos servidores reais para cada serviço virtual.

4.4 Resultados

Os dados abaixo mostram a taxa de utilização do LVS-DR durante quatro dias da fase de teste de produção. Eles explicitam quão poucos pacotes precisam passar pelo diretor devido à particularidade do serviço `xdmcp`. Como a conexão é UDP, não há pacotes de saída.

```
[root@midas:~] # ipvsadm -L --stats
IP Virtual Server version 1.2.1 (size=65536)
Prot LocalAddress:Port          Conns   InPkts  OutPkts  InBytes OutBytes
  -> RemoteAddress:Port
UDP  lvs.ime.usp.br:xdmcp         88       94        0     3290        0
  -> 192.168.45.13:xdmcp          20       21        0      735        0
  -> 192.168.45.43:xdmcp          36       39        0     1365        0
```

5 *Howto*

Neste capítulo mostraremos como implementar e manter a solução utilizando LVS-DR. Para tanto será necessário o pacote que contém o diretório `lvs`, descrito na seção 4.3, e que todas as máquinas sejam acessíveis de uma mesma rede.

Determine os valores do VIP, DIPs, para o diretor e o estepe, e RIPs, para os servidores reais.

5.1 Diretor e estepe

A instalação do diretor e do estepe são idênticas. Neste *howto* usaremos a distribuição Debian[4] e, no caso do uso de outras distribuições, os pacotes utilizados devem ser equivalentes aos mencionados abaixo.

Certifique-se que as máquinas estão com as interfaces de rede configuradas com os respectivos DIPs.

5.1.1 Configurações do *Kernel*

Inicialmente, pegue a versão mais atual do código-fonte do *kernel* 2.6[1] e certifique-se de que as seguintes opções de compilação estejam selecionadas. Esta versão é a mais indicada porque, até a versão 2.4, o LVS era um *patch* e agora é parte integrante do *kernel*.

```
Code maturity level options --->
    [*] Prompt for development and/or incomplete code/drivers

Device Drivers  --->
Networking support  --->
Networking options --->
[*] Network packet filtering (replaces ipchains)  --->
    [ ]   Network packet filtering debugging
    IP: Netfilter Configuration  --->
IP: Virtual Server Configuration  --->
```

[1] Para utilizar versões mais antigas, como a 2.2 ou 2.4, siga as instruções em http://www.linuxvirtualserver.org/VS-DRouting.html

```
<*> IP virtual server support (EXPERIMENTAL)
[*]   IP virtual server debugging
(16)   IPVS connection table size (the Nth power of 2)
---   IPVS transport protocol load balancing support
[*]   TCP load balancing support
[*]   UDP load balancing support
[*]   ESP load balancing support
[*]   AH load balancing support
---   IPVS scheduler
<M>   round-robin scheduling
<M>   weighted round-robin scheduling
<M>   least-connection scheduling scheduling
<M>   weighted least-connection scheduling
<M>   locality-based least-connection scheduling
<M>   locality-based least-connection with replication scheduling
<M>   destination hashing scheduling
<M>   source hashing scheduling
--- IPVS application helper
<M>   FTP protocol helper
```

Compile e instale o novo *kernel*.

5.1.2 Pacotes

Instale os seguintes pacotes, necessários para o funcionamento do LVS.

- `modutils`;
- `make`;
- `gcc`;
- `libncurses5-dev`;
- `ipvsadm`;
- `mon`;
- `heartbeat`;
- `fping`

5.1.3 Configurações

Copie o diretório `lvs` para `/root` e altere o arquivo `/etc/hosts`, incluindo os nomes (igual a `uname -r`) e endereços IP do diretor e do estepe.

Mon

Edite o arquivo `/etc/init.d/mon` e altere os campos `user` e `group`, respectivamente, para `root` e `root`. Isso é necessário para que o `mon` seja capaz de alterar a tabela LVS.

Retire o *script* de inicialização do `mon` do *runlevel default*. O `heartbeat` cuidará de sua ativação quando necessário.

```
# update-rc.d -f mon remove
```

Crie um *link* simbólico para o arquivo `/root/lvs/bin/lvs.alert` no diretório `/usr/lib/mon/alert.d`:

```
# ln -s /root/lvs/bin/lvs.alert /usr/lib/mon/alert.d
```

O arquivo de configurações do `mon` será criado automaticamente mais tarde.

Heartbeat

Para que o `heartbeat` funcione corretamente, suas configurações em ambos o diretor e o estepe devem ser idênticas.

Crie *links* simbólicos em `/etc/ha.d/resources.d` para os *scripts* de inicialização acima:

```
# ln -s /root/lvs/bin/director.lvs.dr /etc/ha.d/resources.d/
# ln -s /etc/init.d/mon /etc/ha.d/resources.d/
```

Altere os seguintes arquivos:

- `/etc/heartbeat/ha.cf`

  Configure as seguintes variáveis:

```
debugfile /var/log/ha-debug   # Arquivo de debug do heartbeat
logfile /var/log/ha-log       # Arquivo de
keepalive     2               # Tempo entre heartbeats
deadtime      30              # Tempo para declarar máquina como morta
warntime      10              # Tempo para declarar suspeita de morte
initdead      120             # Tempo usado para inicialização
udpport       694             # Porta para comunicação
bcast         bond0           # Nome da interface de rede usada na
                              # comunicação
auto_failback legacy          # Garante compatibilidade com versões
                              # antigas
node          midas           # Nome do diretor (igual a uname -r)
node          estepe          # Nome do estepe (igual a uname -r)
```

- `/etc/heartbeat/haresources`

  Este arquivo deve conter apenas uma linha, como o exemplo abaixo. Substitua o nome do diretor pelo apropriado, o VIP e a máscara.

  ```
  midas 192.168.45.26/32 director.lvs.dr mon
  ```

lvs.conf

Configure o arquivo `/root/lvs/etc/lvs.conf` para as propriedade de sua implementação do LVS. Segue um modelo:

```
# Os valores padrão para as 4 primeiras linhas não precisam ser
# alterados.

monfile   /etc/mon/mon.cf      # Arquivo de configuração do mon
cfbasedir /etc/mon             # Diretório base do mon
mondir    /usr/lib/mon/mon.d   # Diretório de monitores do mon
alertdir  /usr/lib/mon/alert.d # Diretório em que se encontra
                               # lvs.alert

lvshost   192.168.45.26        # VIP

# Cada serviço virtual deve seguir o modelo abaixo.
# service <protocolo> <porta> <escalonador> {
#        <RIP 1> <peso>
#        <RIP 2> <peso>
#        ...
#        <RIP n> <peso>
# }
#
# O escalonador pode ser:
# rr, para round robin
# wrr, para weighted round robin
# lc, para least connection
# wlc, para weighted least connection
#
# No caso de escalonador sem peso, <peso> deve ser igual a 1
# para todos os servidores reais.

service udp 177 wrr {
        192.168.45.13 1
        192.168.45.43 1
}
```

Execute `/root/lvs/bin/configure` para criar o arquivo de preenchimento da tabela LVS (`/root/lvs/bin/fill_table`) e escrever o arquivo de configurações do `mon`

(`/etc/mon/mon.cf`).

5.1.4 Balanceamento dinâmico de processamento

Para ativar o balanceamento dinâmico de carga, basta incluir uma linha do *script* `poll.pl` no arquivo `/etc/crontab`. Certifique-se de que os passos relevantes nos servidores reais já foram executados. Caso contrário, ou se algum dos servidores reais não for Linux, esse processo não funcionará corretamente e resultados inesperados podem ocorrer.

```
* * * * * root test -x /root/lvs/util/poll.pl && /root/lvs/util/poll.pl
```

Reinicie as máquinas para que o `heartbeat` inicialize o diretor e o estepe.

5.2 Servidores reais

Configure os serviços nos servidores reais normalmente. Lembre-se de que é necessário que aqueles que oferecerão o mesmo serviço virtual devem ter configurações iguais.

A interface de rede dos servidores reais devem ter um *alias* com o VIP escolhido, para participar do esquema. Para isso, copie o *script* `/root/lvs/util/realserver.lvs.dr`, que está disponível no diretor, para os servidores reais (apenas máquinas Linux) e configure para que seja executado durante a inicialização da máquina.

Além disso, quando um usuário final faz um pedido de conexão ao LVS, é necessário impedir que esse *alias* responda a requisições ARP direcionadas ao VIP; apenas o diretor deve fazer isso. Para isso, em máquinas Linux rodando Debian, deve-se alterar o arquivo `/etc/sysctl.conf`, incluindo (ou alterando) as seguintes linhas no final:

```
# Altere eth1 para o nome da interface de rede usada.

net.ipv4.conf.eth1.arp_ignore = 1
net.ipv4.conf.eth1.arp_annouce = 2
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_annouce = 2
```

5.2.1 Balanceamento dinâmico de processamento

Copie os arquivos `/root/lvs/util/uptimesrv` e `/root/lvs/util/uptimesrv.pl` para os servidores reais. Configure a variável `BASEDIR` do arquivo `uptimesrv` com o diretório

no qual o arquivo `uptimesrv.pl` ficará. Além disso, certifique-se de que `uptimesrv` será executado durante a inicialização.

Inicie os serviços `realserver.lvs.dr` e `uptimesrv`.

6 Conclusão

Redes de computadores estão cada vez maiores e mais complexas. Conseqüentemente, mais frágeis e sujeitas a falhas de todos os tipos, de ataques de pessoas mal intencionadas até falta de energia, de administradores relapsos a manutenção preventiva. A redundância tem como objetivo minimizar as conseqüencias dessas falhas e permitir que os usuários usufruam dos serviços oferecidos a qualquer hora.

O balanceamento de carga e de processamento ajuda a manter o tempo de resposta dos serviços menor, já que cada servidor real recebe menos requisições e tem mais recursos disponíveis para tratar cada uma delas. Como resultado, há um aumento na confiabilidade do sistema e da satisfação dos usuários.

A implementação na rede do IME garantiu maior transparência para os usuários do servidor de aplicação. Embora já houvesse servidores duplicados, enquanto antes os usuários necessitavam escolher qual servidor usar, agora só precisam fornecer sua identificação e senha; as conexões são distribuidas de forma mais equilibrada.

Além disso, disponibilizou uma solução genérica à qual se pode adicionar qualquer serviço futuramente, beneficiando-se não só da redundância como também do balancemanto de carga.

A implementação de redudância pode ser bastante cara por requerer o uso de diversas máquinas. O LVS é uma solução de redundância e balanceamento de ótimo custo/benefício. Ele permite que máquinas de baixo poder de processamento (com 600 a 800 MHz) possam ser empregadas como diretor (ou estepe), garantindo a continuidade dos serviços e o balanceamento da carga.

Vale lembrar que o uso de redundância, mesmo para discos rígidos, não elimina a necessidade de uso de *backup* e de sua atualização constante. Em última instância, seu uso é a única garantia de continuidade da disponibilidade dos serviços.

Parte II

Parte Subjetiva

7 Guilherme Camilo Amantéa

Quando o professor Arnaldo Mandel sugeriu que fizéssemos esta implementação como Trabalho de Conclusão de Curso, fiquei bastante motivado. Afinal, é algo que requer um certo estudo sobre soluções existentes, estudo de viabilidade na rede do IME e, ainda, a implementação. Ou seja, seria algo imediatamente útil e as pessoas usufruiriam diretamente de todos os aspectos, sentindo uma melhora (ou não) de forma bem aparente.

A maior preocupação do professor, desde o início, era fazer um balanceamento de carga mais uniforme entre os servidores de aplicação. Com a chegada de novos servidores, seria desejável que os usuários os utilizassem. Não adiantaria nada investir em novos computadores para serviço de aplicação se todos os usuários sempre escolhessem um ou dois na lista. Daí a necessidade de transparência, que implica em uma melhor distribuição de utilização.

Uma conseqüência gratificante de nossos estudos foi que a solução encontrada não resolve apenas o problema principal, mas pode ser usada, sem alteração na estrutura, para aplicar redundância e balanceamento de carga em qualquer serviço TCP ou UDP. A princípio, pode-se, inclusive, usar servidores reais de qualquer sistema operacional. Como a rede do IME possui alguns servidores que rodam Windows, isso poderá ser útil no futuro.

Um outro aspecto do trabalho que também foi muito gratificante foi que, na tentativa de criar ferramentas para facilitar a administração do sistema implementado, escrevemos alguns *scripts* em `perl`. Eu não sabia programar em `perl`, mas agora me sinto capaz de utilizar essa linguagem para fazer quase qualquer coisa que eu conseguiria usando *bash scripting*, na qual tenho certa experiência, adquirida durante meus dois anos como administrador da Rede GNU/Linux do IME.

7.1 Desafios e frustrações

Existiram, ao longo do ano, algumas surpresas e constatações desagradáveis. Apesar do fato de que, na minha opinião, o balanço geral foi bastante positivo, não se pode esquecer de que nem tudo foi como o imaginado.

Este trabalho exigiu um esforço grande, tanto na fase de estudo quanto na

de implementação, mas, principalmente, na de implementação. Gastamos muito tempo resolvendo problemas técnicos nada interessantes, como placas de rede que paravam de funcionar, detalhes técnicos de instalação de serviços em servidores de testes, dificuldades com dependências de pacotes no Debian, etc.

Apesar do fato de que eu tenho interesse em projetos de redes de computadores, eu já possuía um conhecimento básico sobre o assunto, resultado de minha experiência como administrador da Rede GNU/Linux. Claro, meu conhecimento foi amplamente aumentado durante o trabalho, mas eu senti falta de mais tempo disponível para conhecer melhor algumas áreas acadêmicas, como Teoria dos Grafos e Combinatória.

Uma frustração menos relevante foi que, no início dos testes em laboratório, fiquei empolgado com a idéia de montar um pacote Debian com a nossa solução. Esse assunto ficou em aberto até que o trabalho já estivesse com uma cara mais bem definida. Porém, no final do semestre, o tempo começou a ficar curto para fazer todas as coisas necessárias. A estrutura do trabalho já estava pronta. Já estava até implementada na rede do IME, mas tínhamos que fazer apresentação e monografia, que tomaram bastante do nosso tempo, de forma que não conseguimos mais dar atenção àquele assunto.

7.2 Disciplinas relevantes para o trabalho

Eis uma pequena lista de disciplinas que considero mais diretamente relevantes para este trabalho.

- MAC0110 - Introdução à Computação Nessa disciplina, tive meu primeiro contato com a linguagem Java e, por conseqüência, com as primeiras idéias sobre orientação a objetos, prática que considero muito útil na contribuição à forma como se encara os programas. Além disso, é um dos primeiros contatos do aluno com programação, o que contribui para que ele pense de maneira mais estruturada.

- MAC0122 - Princípios de Desenvolvimento de Algoritmos Essa disciplina aprofunda grandemente a forma estruturada de se pensar quando se quer escrever um programa. Esse pensamento estruturado ajuda o aluno a sempre levar em consideração os casos extremos do problema em questão.

- MAC0323 - Estruturas de Dados Essa disciplina foi extremamente útil por apresentar uma grande variedade de estruturas de dados, que são conceitos abstratos que

podem ser aplicados a muitos problemas. Usamos vários deles na implementação dos *scripts* de administração do nosso projeto.

- MAC0211 - Laboratório de Programação I Essa disciplina nos apresentou várias ferramentas muito úteis, algumas das quais foram utilizadas no nosso trabalho, como LaTeX, `convert`, `xfig`, `make`, etc.

- MAC0338 - Análise de Algoritmos Essa disciplina nos mostra mais claramente a necessidade de preocupação com a eficiência dos algoritmos utilizados. A partir dela, temos mais recursos para implementar algoritmos eficientes e verificar matematicamente qual sua complexidade. Ela é útil para qualquer coisa que envolva fazer algoritmos.

- PCS0210 - Redes de Computadores Apesar de não muito aprofundada, essa disciplina me apresentou alguns conceitos que eu não conhecia, como o funcionamento dos protocolos TCP e UDP, as estruturas de camadas OSI e TCP/IP, endereço IP, máscara de rede, etc.

7.3 Aprofundamento

Não é nula a possibilidade de que meu trabalho seja na área de projetos de redes de computadores. Para isso, porém, há uma necessidade de revisar conceitos teóricos apresentados, principalmente, na disciplina PCS0210 e, também, pesquisar sobre soluções utilizadas em grandes empresas para ganhar experiência e aumentar meu repertório de ferramentas disponíveis para resolver problemas de clientes.

Meu conhecimento sobre redundância é bastante restrito ao que foi estudado para este trabalho. Este apresenta uma solução geral para redundância de servidores, mas contém algumas pequenas imperfeições. Por exemplo, quando o diretor, por algum motivo, cai, o estepe demora 10 segundos para perceber o ocorrido e assumir seu lugar. Apesar de esse tempo ser configurável, eu gostaria de pesquisar formas de que seja mais dinâmico. Quem sabe até garantir que, se algum usuário estiver usando um servidor real e este cai, aquele seja transferido para outro sem perceber o que aconteceu. Na nossa solução, se isso acontece, o usuário terá que se conectar novamente para continuar usando o serviço.

Além disso, eu também gostaria de pesquisar sobre outras formas de redundância

implementadas em grandes empresas para compará-las e saber as vantagens e desvantagens de cada uma. Quem sabe até inventar uma nova forma melhor do que a que existe atualmente para algum contexto específico?

Também há o interesse em estudar formas de redundância não necessariamente aplicadas a servidores e clientes numa rede de computadores. Provavelmente, conceitos aplicados em redundância de componentes de hardware ou mesmo fora do contexto de computadores podem ser aplicados para melhorar soluções já existentes. Além disso, eu gostaria de ser capaz de abstrair as características relevantes das melhores soluções para poder ser capaz de implementá-las em qualquer contexto, relacionado a computadores ou não.

8 Guilherme Grimaldi Nepomuceno

Até o início das aulas de 2007, não havia decidido ainda qual seria a área a que me dedicaria em meu trabalho de formatura supervisionado. Foi então que o Guilherme Amantéa me convidou para trabalhar junto a ele em um projeto do professor Arnaldo Mandel para a rede de computadores do IME.

A idéia de um projeto real, que poderia beneficiar várias pessoas me pareceu bastante interessante. Como o projeto envolvia redes de computadores, uma área na qual eu já possuía algum conhecimento por trabalhar em uma empresa do setor, achei o convite ideal e o aceitei.

Foi uma experiência muito proveitosa. Tive oportunidade de trabalhar em equipe em um projeto maior e mais longo, além de ter contato com administradores de rede. Aprendi, na prática, *bash script* e *perl* para implementar as rotinas de redundância e manipulações de arquivos. Até mesmo as dificuldades que encontramos serviram para mostrar como um projeto real se comporta.

8.1 Desafios e Frustrações

Quando iniciamos o projeto, tínhamos quatro objetivos principais: entender o funcionamento da rede do IME, assim como descobrir quais eram as máquinas servidores e quais serviços estavam disponíveis e onde; verificar qual era a real necessidade de redundância e pra quais serviços e implementá-la; melhorar o esquema de religamento de servidores caídos; e controlar melhor o uso dos servidores de aplicação, já replicados.

Vimos que a rede do IME era grande e complexa e que nem mesmo os próprios administradores tinham uma relação pronta do que rodava em cada servidor. Foi então que percebemos que seria extremamente complicado tentar tratar o problema caso a caso, ou melhor, serviço a serviço. Precisávamos de uma solução simples e unificada, confiável e robusta. Foi então que descobrimos o LVS.

Ele parecia ser ideal para nosso propósito: servia para qualquer serviço, era capaz de lidar com qualquer número de servidores e ainda oferecia balanceamento de carga, além de remover a necessidade do esquema de religamento. Seu único problema

era sua documentação; ela não era objetiva e estava desatualizada (usava a versão 2.4 do *kernel*, enquanto nós usamos a versão 2.6). Porém era bastante completa, e depois de muita leitura, tínhamos escolhido o modelo de redundância que usaríamos.

Nosso próximo desafio foi testar o modelo. Conseguimos com a SI quatro máquinas antigas, porém não tínhamos onde colocá-las. Felizmente, devido a problemas com algumas máquinas da Rede Linux, conseguimos um espaço na sala 258A, onde pudemos realizar nossos testes.

A partir de então o trabalho realmente andou, apesar dos insistentes pequenos problemas com as máquinas de testes, que, pelo menos, serviram para testar a robustez da solução e mostrar o que deveríamos melhorar.

Por fim, refinamos a solução, incluindo monitoração, balanceamento de processamento e nossos próprios *scripts* de administração. Tentamos criar uma solução genérica bastante eficiente, adaptável e de fácil manutenção.

8.2 Disciplinas relevantes para o trabalho

De certa forma, acredito que todas as disciplinas cursadas foram importantes de alguma forma. Abaixo segue uma lista das matérias que acredito terem mais contribuído no decorrer deste projeto:

- MAC0110 - Introdução à Computação

  Embora eu tenha tido essa matéria há muito tempo, ela serviu de base para todo o curso. Foi dada em Pascal, mas serviu para fixar os conceitos mais básicos que utilizo sempre.

- MAC0122 - Princípios de Desenvolvimento de Algoritmos

  Aqui, finalmente tive contato com técnicas mais elaboradas para soluções de problemas. Foi a matéria que mais me motivou no curso e onde eu finalmente aprendi a programar em C.

- MAC0323 - Estruturas de Dados

  Os *scripts* que criamos para administrar nossa solução utilizam várias estruturas de dados complexas, como tabelas de *hashing*, para manipulação dos dados. Essa disciplina foi essencial para seu entendimento e uso correto.

- MAC0211 - Laboratório de Programação I

  Para mim, essa disciplina foi muito importante, pois foi meu primeiro contato mais profundo com o ambiente Linux, assim como suas ferramentas e documentação, *man pages*, que foram muito utilizadas durante o projeto.

- MAC0422 - Sistemas Operacionais

  Essa disciplina me apresentou o funcionamento do módulo mais básico de um computador, seu *kernel*, e mostrou como customizá-lo e compilá-lo. Foi muito útil no projeto pois o LVS é um módulo de *kernel*.

- PCS0210 - Redes de Computadores

  Essa disciplina foi muito importante por expor os conceitos com que trabalhamos no projeto, como o protocolo TCP e o UDP, além de alguns serviços, como DNS e e-mail, e da própria necessidade de redundância.

- EAD0610 - Introdução a Administração

  Apesar de ser uma disciplina fora de área e tê-la cursado no último semestre, ela me proporcionou algumas dicas de como melhor administrar meu tempo entre estudos, provas e o projeto.

8.3 Considerações Finais

A meu ver, o projeto ainda não chegou a seu fim. Um próximo passo lógico seria a reestruturação da rede, melhor aproveitando os servidores e a estrutura do LVS e da redundância, assim como uma integração com o sistema Windows. Espero que o texto que preparamos possa ajudar os administradores da rede do IME a manter a solução e aplicá-la onde mais for necessário.

Referências Bibliográficas

[1] Randell, B.; Lee, P. A. e Treleaven, P. C., *Reliability Issues in Computing System Design*, ACM Computing Surveys, Volume 10, Issue 2 (June 1978)

[2] The Linux Virtual Server Project, http://www.linuxvirtualserver.org

[3] O'Rourke, Patrick e Keefe, Mike, *Performance Evaluation of Linux Virtual Server* http://www.linuxvirtualserver.org/performance/lvs.ps.gz

[4] Debian, http://www.debian.org

[5] Wikipedia, http://www.wikipedia.org

[6] The Linux Documentation Project, http://tldp.org, para consulta de *man pages* (`bash`, `ipvsadm`, `mon`, `heartbeat`)

[7] Newbam, Cameron e Rosenblatt, Bill, *Learning the bash shell*, O' Reilly, 3ª Edição

[8] Wall, Larry; Christiansen, Tom e Orwant, Jon, *Programming Perl*, O' Reilly, 3ª Edição