Wanderson Santiago dos Reis

Virtualização de serviços baseado em contêineres: Uma Proposta para Alta Disponibilidade de Serviços em Redes Linux de pequeno porte

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Me. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2009

Wanderson Santiago dos Reis

Virtualização de serviços baseado em contêineres: Uma Proposta para Alta Disponibilidade de Serviços em Redes Linux de pequeno porte

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *19 de abril de 2009*

Prof. Me. Herlon Ayres de Camargo

Profa. Dra. Marluce Rodrigues Pereira

Prof. Me. Joaquim Quinteiro Uchôa
(Orientador)

Lavras
Minas Gerais - Brasil
2009

Agradecimentos

À minha mãe por sua dedicação a minha formação.
À minha esposa e filho por suas paciências involuntárias durante as minhas longas horas de trabalho árduo.
À toda equipe do curso ARL por seu brilhante trabalho no contexto nacional.
A todos que contribuiram de alguma forma, muito obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Neste trabalho apresentamos e implementamos uma proposta de solução de baixo custo para tolerância a falhas de *hardware*, de sistema e de comunicação utilizando virtualização de servidores baseado em contêineres GNU/Linux e a solução Heartbeat para *clusters* de alta disponibilidade. A viabilização desta proposta foi possível devido ao emprego e combinação de alguns sistemas FLOSS (*Free/Libre and Open Source Software*) que proporcionam um ambiente altamente estável para aplicações essenciais. A implementação da proposta demonstrou um tempo de resposta adequado em situações simuladas de falha que ativaram o *failover*. Outras vantagens desta estrutura foram percebidas como a otimização do uso do *hardware*, mesmo utilizando componentes redundantes, agregou benefícios com o uso do *software* livre como baixo custo de aquisição e implementação e uso de tecnologias inovadoras na criação de soluções de alta disponibilidade.

Palavras-Chave: alta disponibilidade; virtualização; failover; Heartbeat.

Capítulo 1

Introdução

Apresentamos neste trabalho uma proposta e implementação de uma solução de baixo custo para tolerância a falhas de *hardware*, de sistema e de comunicação utilizando virtualização de servidores baseado em contêineres GNU/Linux e a solução Heartbeat para *clusters* de alta disponibilidade. Todos os aplicativos utilizados no laboratório de implementação são aplicativos livres. O objetivo principal desta proposta é manter a confiança e disponibilidade de serviços essenciais de redes nos ambientes de pequenas e médias empresas, sem aquisição de *hardware* especializado.

A motivação do trabalho é dada pela crescente demanda por serviços *on-line* que devem estar sempre disponíveis, pois interferem na vida de muitas pessoas que dependem diretamente e indiretamente de tais serviços. A demanda por disponibilidade de serviços atinge a todos, sejam as pessoas ou as empresas independente de seu porte. A virtualização de servidores como uma técnica de gerenciamento de serviços, de compartilhamento eficiente de recursos computacionais e de redução de custos, vem sendo cada vez mais aplicada por grandes empresas. Atualmente o custo e a complexidade não são mais fatores impeditivos de acesso às tecnologias de virtualização e de *clusters* de alta disponibilidade. A implementação destas técnicas e sua aplicação pode ser feita por qualquer empresa ou pessoa, mesmo em ambiente doméstico. O principal motivo deste novo cenário é a disponibilidade de uma grande variedade de sistemas FLOSS (*Free/Libre and Open Source Software*) que combinados proporcionam um ambiente altamente estável para aplicações essenciais com impacto direto na produtividade e no faturamento das empresas.

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta brevemente as técnicas de virtualização e em especial a virtualização baseada em

contêineres; o Capítulo 3 se dedica a mostrar os principais conceitos relacionados a alta disponibilidade e justifica o uso de uma arquitetura de *cluster* para a alta disponibilidade; o Capítulo 4 apresenta a proposta de integração das tecnologias tratadas anteriormente; o Capítulo 5 apresenta detalhes técnicos da implementação da proposta; o Capítulo 6 valida a proposta demonstrando o comportamento do ambiente durante simulação de falhas e o Capítulo 7 apresenta as considerações finais e propostas de trabalhos futuros.

Capítulo 2

Virtualização

A virtualização é uma tecnologia que oferece uma camada de abstração dos verdadeiros recursos de uma máquina, provendo um *hardware* virtual para cada sistema (SILVA, 2007). Apesar de qualquer sistema computacional possuir um caráter virtual em relação ao mundo em que vivemos, ainda é possível aplicarmos técnicas na virtualização de sistemas operacionais.

A virtualização traz benefícios, sob o ponto de vista dos negócios, pois diminue os custos com *hardware* (consolidação), otimiza a distribuição de carga dos serviços e flexibiliza o gerenciamento dos recursos (IBM, 2005). A imediata conseqüência da virtualização de servidores é a economia com energia, espaço, ambientação para um menor número de máquinas e a mão-de-obra com sua administração. Um ponto importante destacado por (JONES, 2006) é a possibilidade de migração de um sistema operacional e suas aplicações para um novo *hardware* para balancear a sobrecarga do mesmo sem prejuízo na disponibilidade do serviço.

A virtualização de sistemas operacionais emprega diversas técnicas distintas desde a emulação completa do *hardware* até a virtualização no nível do sistema operacional, também conhecida como *Container-based Operating System Virtualization* ou virtualização de sistema operacional baseada em contêineres. Neste capítulo vamos tratar brevemente as principais técnicas de virtualização e em especial a virtualização baseada em contêineres implementada pelo Linux-VServer.

2.1 Técnicas de virtualização

A virtualização de sistemas operacionais utiliza técnicas testadas e aperfeiçoadas desde os anos sessenta quando a empresa IBM começou a desenvolver a primeira máquina virtual (MATTOS, 2008). Esta secção apresenta brevemente as técnicas de virtualização mais utilizadas em ambientes de produção, são elas emulação de *harware*, virtualização completa, paravirtualização e virtualização baseada em contêineres.

A emulação de *hardware* é uma das técnicas mais antigas de virtualização. Esta técnica cria uma camada completa de abstração de *hardware* sobre a qual o sistema operacional será executado. A técnica de emulação de *hardware* é importante para desenvolvedores de *firmware* e de *hardware*, pois permite simular e validar aplicativos sem a presença do *hardware* real (OLIVEIRA, 2007). A maior desvantagem se refere a baixa performace do conjunto emulado. O QEmu[1] e o Bochs[2] são implementações conhecidas da técnica de emulação de *hardware*.

A virtualização completa ou nativa emprega uma máquina virtual que faz a comunicação entre o *hardware* e o sistema virtualizado. O papel da máquinal virtual é compartilhar o *hardware* de forma harmônica entre o sistema operacional hóspede e o hospedeiro. Segundo (JONES, 2006) está técnica apresenta uma melhor performance comparada a emulação de *hardware*, mas ainda menor se comparada ao uso nativo do *hardware*. A principal vantagem da virtualização completa é a possiblidade de virtualização de sistemas operacionais não modificados, igualmente na emulação, contudo neste caso o sistema virtualizado deve suportar o *hardware* da máquina onde a virtualização completa for empregada. O VMWare[3], VirtualBox[4] e o KVM[5] (*Kernel-based Virtual Machine*) são exemplos de produtos que utilizam a técnica de virtualização completa. O KVM vem se destacando recentemente pois foi incluído oficialmente na árvore de desenvolvimento do *kernel* do Linux (apartir da versão 2.6.20). O VirtualBox e KVM já suportam o uso de extensões do processador (Intel VT ou AMD-V) que porporciona um melhor aproveitamento do *hardware* com impacto no desempenho das máquinas virtuais.

A paravirtualização emprega o uso de uma máquina virtual, chamada de hipervisor, que compartilha o acesso ao *hardware* físico da máquina hospedeira. A principal diferença em relação a virtualização completa é a necessidade de alteração

[1] `http://bellard.org/qemu/`.
[2] `http://bochs.sourceforge.net/`.
[3] `http://www.vmware.com/`.
[4] `http://www.virtualbox.org/`.
[5] `http://www.linux-kvm.org`.

dos sistemas hóspede e hospedeiro. A alteração é necessária pois os sistemas hóspede e hospedeiro cooperam entre si durante o processo de virtualização através do hipervisor. Segundo (NEVES; PACHECO, 2008) embora a paravirtualização apresente um ganho de desempenho significativo frente à virtualização completa, esta vantagem tem sido superada devido à presença de instruções de virtualização nos processadores Intel e AMD, que favorecem a virtualização completa. O Xen[6] Hypervisor e o UML[7] são exemplos de implementação da paravirtualização.

A virtualização baseada em contêineres ou também chamada de virtualização a nível de sistema operacional utiliza uma camada às chamadas do sistema (*system calls*) o que permite o isolamento entre o sistema hospedeiro e a máquina virtual. O principal benefício da virtualização baseada em contêiners é uma performance nativa do sistema e a desvantagem é o uso de somente um tipo de sistema operacional (JONES, 2006).

Segundo (SOLTESZ; POTZL; FIUCZYNSKI, 2007) as técnicas de virtualização de sistemas operacionais baseadas em contêineres chegam a uma performace até duas vezes superior às outras técnicas, principalmente em cenários de serviços baseados em web. De acordo com (PADALA *et al.*, 2007) a peformance de sistemas paravirtualizados piora no momento em que se aumenta a quantidade de aplicações estanciadas de uma para quatro. Neste cenário o tempo médio de resposta dos servidores paravirtualizados aumenta quatro vezes enquanto que usando virtualização baseada em contêineres este tempo de resposta somente dobra. Com base neste estudo observa-se que a técnica de virtualização baseada em contêineres apresenta um melhor desempenho quando se demanda mais máquinas virtuais rodando simultaneamente. O OpenVZ[8] e o Linux-VServer[9], objeto do nosso estudo, são exemplos de implementação da virtualização baseada em contêineres para sistemas Linux.

2.2 Linux-VServer

O Linux-VServer é uma das principais implementações da técnica de virtualização baseada em contêineres. Esta implementação virtualiza um *kernel* do Linux modificado o que permite a existência de múltiplos espaços de usuário. Estes espaços de usuário coexistem de forma isolada, sem interferência entre si. O Linux-VServer

[6] `http://www.xen.org/`.

[7] ***User Mode Linux*** `http://user-mode-linux.sourceforge.net/`.

[8] `http://wiki.openvz.org/`.

[9] `http://linux-vserver.org/`.

emprega diversas alterações no código do *kernel* do Linux para promover o isolamento entre os VPS (*Virtual Private Servers*).

O isolamento entre os diversos espaços do usuário é baseado no conceito de contexto. Cada contexto é um contêiner para os processos de uma determinada máquina virtual, desta forma as ferramentas de usuário somente atuam nos processos de seu próprio contexto. Durante a inicialização o *kernel* cria um contexto padrão. Para o administrador fica disponível um visualizador de contextos que permite listar os processos de todos os contextos.

O Linux-VServer emprega a técnica de enjaulamento (*chroot*) para isolar o diretório raiz de cada máquina virtual. Uma jaula cria um novo diretório raiz, contudo um recurso especial chamado de *chroot-barrier* não permite que uma máquina virtual acesse um diretório raiz de outro contexto usando a mesma técnica de enjaulamento. A partir de cada diretório raiz, devidamente isolado, cada contexto terá sua própria lista de usuários e senha de super usuário.

O Linux-VServer pode ser utilizado com o *kernel* do Linux nas versões 2.4 e 2.6, desde que seja modificado com a aplicação de *patches*, pode ser executado em diversas plataforma de *hardware* como x86, x86-64, SPARC, MIPS, ARM e PowerPC. Segundo (SOLTESZ; POTZL; FIUCZYNSKI, 2007) e de acordo com seus experimentos a eficiência do Linux-VServer é sempre compatível com a utilização de um *kernel* do Linux não virtualizado. Na Tabela 2.1 podemos observar os tempos de resposta do sistema em um ambiente nativo rodando um *kernel* do Linux sem modificação, um *kernel* do Linux-VServer e um *kernel* do Linux Xen3.

Tabela 2.1: *Benchmark* em microsegundos do LMbench OS para *kernel* monoprocessados (quanto menor melhor)

Parâmetro	*kernel* do Linux	*kernel* do Linux-VServer	*kernel* do Linux Xen3
fork process	103,2	103,4	282,7
exec process	252,9	253,6	734,4
sh process	1054,9	1047,1	1816,3
ctx (2p/ 0K)	2,254	2,496	3,549
ctx (16p/16K)	3,008	3,310	4,133
ctx (16p/64K)	5,026	4,994	6,354
page fault	1,065	1,070	2,245

Fonte: (SOLTESZ; POTZL; FIUCZYNSKI, 2007)

Capítulo 3

Alta Disponibilidade

Sistemas de alta disponibilidade apresentam técnicas que visam a continuidade do serviço (WEBER, 2002). Podemos concluir que alta disponibilidade significa manter algo disponível durante o maior período de tempo possível. No ambiente de sistemas computacionais a alta disponibilidade quase sempre implica em redundâncias de *hardware* e *software*. Manter um sistema sem interrupção é uma tarefa que demanda altos investimentos. Estes custos podem e devem ser compensados à medida que o prejuízo causado para os usuários ou para os negócios do usuário alcançam proporções intoleráveis.

Soluções de alta disponibilidade para sistemas utilizam diversas técnicas para minimizar os impactos de uma possível falha em diversos pontos do sistema. Soluções de alta disponibilidade geralmente são avaliadas com base em níveis de disponibilidade, segundo o cálculo de tempos de indisponibilidade ou *downtime*. Geralmente, quanto maior a disponibilidade, maior a redundância e o custo das soluções (PEREIRA, 2005). Na Tabela 3.1 são exibidos os valores calculados para cada nível de disponibilidade, estes índices são frequentemente utilizados em Acordos de Nível de Serviços ou SLAs (*Service Level Agreement*) nos contratos de terceirização de infraestrutura. É importante observar que o tempo de *downtime* não considera as paradas programadas para manutenção dos sistemas. Neste capítulo abordaremos as principais técnicas computacionais de tolerância à falhas com vistas à manutenção de um sistema íntegro e disponível. Ainda dentro deste tópico são apontadas as principais tecnologias e ferramentas utilizadas na proposta deste trabalho.

Tabela 3.1: Níveis de disponibilidade usados em SLAs

Disponibilidade (%)	*Downtime* anual
99 %	87,5 horas/ano
99.9 %	8,76 horas/ano
99.99 %	52,6 minutos/ano
99.999 %	5,26 minutos/ano

Fonte: (IKE, 2008)

3.1 Técnicas de Tolerância à Falhas

Tolerância à falha é a capacidade de certos sistemas continuarem em operação mesmos em condições de falha. Segundo (PIEDAD; HAWKINS, 2000) os sistemas tolerantes à falhas geralmente são projetados para responderem pró-ativamente a três tipos de erros:

Tempo de resposta - este erro ocorre quanto determinado componente de *software* ou *hardware* demora um tempo superior ao previsto para responder;

Resposta errada - erro típico quando algum componente do sistema retorna uma resposta errada quando se esperava uma resposta correta a partir do emprego de parâmetros conhecidos;

Recurso indisponível - erro apresentado quando algum recurso fica indisponível ou *offline* quando deveria estar operacional.

Para a implentação de técnicas de tolerância a falhas podemos planejar o emprego de estratégias técnicas e tecnológicas para mantermos os sistemas responsivos, íntegros e disponíveis. As técnicas de tolerância a falhas utilizam três tipos de redundâncias:

Hardware - é quando se emprega *harware* especial e/ou componentes de *hardware* duplicados que entram em operação em caso de falha ou trabalham em paralelo. Exemplos de redundâncias de *hardware* são os espelhamentos de discos, fontes de alimentação redundantes, placas de rede espelhadas, módulos de memória com suporte à ECC (*Error Checking and Correcting*), etc;

Sofware - é a manutenção de pelo menos uma cópia exata e operacional do sistema e dos dados em produção. Exemplos de redundância de *software* são

espelhamento de base de dados, de aplicações e demais conteúdos digitais disponibilizados como serviço aos usuários;

Ambiente - é quando se emprega a redundância em recursos necessários ao ambiente como climatização e suprimento de energia elétrica regulada de forma ininterrupta.

Durante o planejamento de soluções de alta disponibilidade cada fator ou ponto de falha poderá ter maior ou menor importância dependendo do contexto de cada negócio. É aconselhável sempre utilizar componentes redundantes de mesmo fabricante / modelo para que durante uma eventual falha a transição de operação entre um componente e outro seja transparente e não crie nenhum ônus operacional. De acordo com (PIEDAD; HAWKINS, 2000) o planejamento e gerenciamento de sistemas de alta disponibilidade devem considerar alguns fatores críticos para o sucesso da solução que é a identificação rápida e imediata correção do ponto falho, pois soluções de alta disponibilidade geralmente são empregados como resposta temporária as ocorrências de falha. Caso um sistema falho não receba a pronta atenção e correção do problema o mesmo ficará sujeito a uma falha permanente e com um *downtime* superior ao planejado / desejado.

O surgimento do sistema operacional GNU/Linux em 1991 impulsionou a criação de diversos serviços e tecnologias FLOSS (*Free/Libre and Open Source Software*) para computação de alto desempenho[1]. Este novo modelo de licenciamento e produção de *software* permitiu a utilização prática de conceitos de computação distribuída (*clustering*) e virtualização (veja Capítulo 2), tanto no ambiente empresarial quanto doméstico.

No contexto da tolerância a falhas, os *clusters* são utilizados como sendo um conjunto de *hosts* ou nós que trabalham compartilhando recursos e com capacidade de redundância. A redundância em *clustering* é caracterizada pela manutenção da disponibilidade do sistema e/ou serviço mesmo que um ou mais nós fiquem indisponíveis (*off-line*). Neste caso outro nó do *cluster* assume as funções do *host* indisponível de forma transparente. Da mesma forma quando o nó indisponível volta a responder ao *cluster* o serviço poderá ser transferido novamente do nó temporário ao nó responsável pelo serviço.

A utilização de arquitetura de *cluster* com nós que repondam por um serviço em caso de falha do nó principal pode ser considerada atualmente uma técnica

[1] Estatísticas do sítio `http://www.top500.org/` apontam que o sistema operacional GNU/Linux é utilizado em 87,8 % dos 500 supercomputadores com mais poder de processamento do mundo, em novembro de 2008.

de tolerância à falha de custo mais acessível. O uso de *cluster* para alta disponibilidade atende plenamente os principais requisitos para um sistema tolerante à falha, pois atende prontamente erros de tempo de resposta e erros de indisponibilidade de recursos. O *cluster*, por sua natureza também implementa redundância de *hardware* e *software*. De acordo com (GOVERNO BRASILEIRO, 2007) a computação distribuída ou *clustering* baseada em *software* livre apresenta as seguintes características:

Baixo custo de implantação;

Independência de fornecedores - facilidade de negociação;

Utilização de *hardware* comum padrão PC;

Baixo custo de manutenção;

Facilidade de redimensionamento do ambiente;

Baixo custo total de propriedade;

Tecnologia inovadora.

3.2 Heartbeat

O Heartbeat é o principal componente do projeto *Linux High Availability Project*[2] que desenvolve aplicativos para soluções de alta disponibilidade para diversas plataformas de sistemas operacionais. Segundo estimativas o Heartbeat é utilizado em mais de trinta mil soluções reais de missão crítica em todo o mundo desde 1999, quando o projeto iniciou (LINUX-HA PROJECT, 2007).

O Heartbeat é um componente de *software* que envia pacotes através de uma conexão física pela rede para outras instâncias do Heartbeat informando que o sistema está ativo. Quando estes pacotes (ou batidas de coração) não são mais recebidas o Heartbeat assume que o nó que parou de enviar os pulsos está indisponível ou "morto". Ao detectar tal situação o Heartbeat executa o procedimentos de *failover* dos recursos e serviços que serão assumidos automaticamente por outro nó da rede. Da mesma forma, quando o nó que estava indisponível fica ativo e responsivo na rede o Heartbeat executa os procedimentos de *failback*, ou seja, restaura os recursos e serviços para o servidor principal novamente. O Heartbeat

[2] http://www.linux-ha.org/

atua como um programa que monitora, toma a decisão e executa os procedimentos tanto para o *failover* quanto para o *failback*. No nosso contexto, utilizamos o Heartbeat para a implementação de um mini *cluster* de alta disponibilidade, maiores detalhes serão apresentados no Capítulo 5.

3.3 DRBD (*Distributed Replicated Block Device*)

O armazenamento redundante no contexto deste trabalho se refere as diversas técnicas de se fornecer capacidade de armazenamento através de uma rede de dados. Armazenamento redundante são técnicas que empregam *hardware* ou *software* que permitem o compartilhamento e o acesso *on-line* dos dados. Segundo (GOVERNO BRASILEIRO, 2007) as técnicas de armazenamento *on-line* utilizam diversas tecnologias como: Disco Rígido, *Storage Devices*, sistemas de arquivos distribuídos, sistemas de arquivos paralelos, dispositivos RAID, controladoras de discos, entre outras.

Em um ambiente de *cluster* de alta disponibilidade quando um nó secundário assume os recursos e serviços do nó primário os dados utilizados pelos serviços devem estar atualizados, íntegros e disponíveis independente do nó primário estar ativo. Desta forma, fica clara a necessidade de um local central onde estes dados fiquem acessíveis para qualquer nó. Este recurso é facilmente implementado através de um disco compartilhado. Segundo (Ellenberg, L., 2007) somente um disco compartilhado não resolve o problema, pois continua a existir um ponto único de falha e a melhor solução é mantermos alta disponibilidade para os dados. Neste ponto é que precisamos de um sistema de armazenamento que forneça a infraestrutura necessária à redundância dos dados.

O DRBD[3] ou Dispositivo de Bloco Replicado e Distribuído, em português, é um aplicativo que permite o espelhamento de dados de dispositivos de blocos (disco rígidos, partições, volumes lógicos, etc) entre nós de uma rede. O DRBD é desenvolvido e fornecido pela empresa Linbit[4] sob os termos da licença GPL, ou seja é um *software* livre e de código aberto.

Segundo (HAAS; REISNER; ELLENBERG,) ele apresenta as seguintes características:

[3] http://www.drbd.org/
[4] http://www.linbit.com/

Replica os dados em tempo real, ou seja, a replicação ocorre de forma contínua enquanto as aplicações alteram os dados no dispositivo;

A replicação é transparente. As aplicações que acessam os dados desconhecem o fato de que os dados estão sendo armazenados em outros dispositivos remotos;

A replicação pode ser síncrona ou assíncrona. No espelhamento síncrono a aplicação que acessa os dados somente recebe a informação que a gravação no disco terminou quando todos os nós tenham feito o mesmo. No espelhamento assíncrono a aplicação que acessa os dados recebe a informação que a gravação no disco terminou quando os dados são gravados localmente, a propagação dos dados para os outros nós é feita posteriomente.

Tecnicamente, o DRBD é um módulo do *kernel* do Linux que implementa um driver para dispositivos de blocos virtuais próprio do *kernel* do Linux. Por sua integração com o *kernel* do Linux, o DRBD apresenta grande flexibilidade e versatilidade ao trabalhar bem próximo da camada de E/S do sistema operacional. A forma de acesso mais utilizada para os dispositivos DRBD é manter somente um disco primário com acesso de leitura-escrita e os demais são secundários[5]. A leitura e escrita dos dados é realizada somente no disco primário e propagadas em tempo real para os discos secundários, em caso de falha do disco primário um dos discos secundários poderá ser promovido a disco primário, mantendo a integridade e disponibilidade dos dados. As transferências dos dados são realizadas sobre o protocolo TCP/IP, por isso esta técnica é também chamada de *RAID 1*[6] *Over Ethernet*.

O DRBD não necessita de nenhum *hardware* especializado e como utiliza os recursos das redes TCP/IP é uma solução com um bom custo / benefício em relação as interfaces e dispositivos dedicados de armazenamento como *Fibre Channel*[7], por exemplo.

[5] O acesso primário-primário é possível mas é dependente de um sistema de arquivos que consiga trabalhar com tal recurso.

[6] RAID 1 é uma técnica onde dois ou mais discos são espelhados e percebidos como somente um disco para o sistema operacional, os dados são gravados simultaneamente em todos os disco e caso um disco falhe os dados ainda continuam disponíveis e íntegros a partir do outro disco.

[7] Fibre Channel ou FC é um padrão definido pelo ANSI em 1994, como uma interface de transferência de dados, visando oferecer uma alternativa para substituição das interfaces de armazenamento de alta velocidade.

Capítulo 4

Metodologia

Nos capítulos 2 e 3, foram tratadas virtualização e alta disponibilidade, respectivamente. Ao final de cada tema apresentamos as ferramentas e tecnologias que integram esta proposta de trabalho, Linux-VServer, Heartbeat e DRBD (*Distributed Replicated Block Device*). Neste capítulo apresentamos o ambiente proposto e seu planejamento.

4.1 Ambiente Proposto

A demanda por serviços de rede é crescente e isso interfere na vida de muitas pessoas que dependem diretamente de tais serviços. A virtualização de servidores vem sendo cada vez mais aplicada por grandes empresas como consequência das vantagens expostas no Capítulo 2. Além disso, a demanda por disponibilidade de serviços atingem a todos, sejam as pessoas ou as empresas independente de seu porte. É importante observar que a demanda por prover serviços de rede que sejam estáveis, responsivos, íntegros e disponíveis não é somente uma necessidade das grandes empresas. Empresas de pequeno e médio porte, com suas pequenas redes de dados e serviços, também demandam tal qualidade em seus serviços de rede. A grande diferença está na economia de escala, pois grandes empresas podem contratar os melhores serviços de redes, investir em equipamentos especializados e nos *links* de dados mais rápidos, por preços competitivos se comparados com seu faturamento, por outro lado as pequenas empresas não conseguem cobrir tais custos. O custo de contratação de somente um servidor dedicado econômico, com

limitados recursos de processamento, memória e armazenamento, fica em torno de 170 a 210[1] dólares por mês.

Atualmente, o acesso às tecnologias de virtualização e de *clusters* de alta disponibilidade não é mais determinado por fatores como o custo, a complexidade e a necessidade de *hardware* especializado. A implementação e aplicação de tais tecnologias podem ser realizadas por qualquer empresa ou pessoa inclusive em ambientes domésticos. O principal motivo deste novo cenário é a disponibilidade de uma grande variedade de sistemas e tecnologias FLOSS (*Free/Libre and Open Source Software*) que combinadas podem proporcionar um ambiente ideal para aplicações essenciais tendo um impacto direto na produtividade, no controle produtivo e no faturamento das empresas de pequeno porte.

No ambiente de trabalho proposto foram integrados:

O Linux-VServer para virtualização de servidores, procurando isolar os serviços e distribuir os recursos de forma mais adequada;

O DRBD para a implementação de um armazenamento redundante de dados, através do espelhamento de discos via rede dedicada;

O Heartbeat para o monitoramento e controle da transferência dos servidores virtuais entre os nós de forma transparente.

Para o nossa proposta consideramos as redes Linux de pequeno porte. Uma rede Linux de pequeno porte no contexto do nosso trabalho significa uma rede que demanda poucos serviços de rede ativos, entre dois e quatro serviços. Serviços típicos em Redes Linux são compartilhamento de arquivos, acesso à internet, LDAP, Banco de Dados, servidor HTTP, servidor DNS, servidor FTP, entre outros.

4.2 Arranjo físico e lógico

Um dos pontos chave em um sistema com alta disponibilidade é a redundância. Em nossa proposta conseguimos atender o requisito de redundância empregando um arranjo físico e lógico de baixo custo. O arranjo físico conta somente com um disco rígido e uma interface de rede adicional, ambos dispositivos de baixo custo atualmente. O disco adicional mantém o espelhamento do servidor Linux-VServer

[1] Empresas consultadas `http://www.maxihost.com.br/`, `http://www.hostlocation.com.br/`, `http://braslink.com/` e `http://www.uolhost.com.br/` em novembro 2008 .

e o dos seus respectivos dados, comparativamente tal disco adicional naturalmente já poderia ser empregado em um espelhamento RAID 1 convencional.

A estratégia de nossa proposta foi distribuir a carga dos serviços de rede entre dois servidores físicos. Cada servidor físico instalado com um servidor Linux-VServer que mantém os serviços de rede. Os dois servidores permanecem interligados por uma interface de rede dedicada que permite o monitoramento mútuo entre os dois *hosts* e o espelhamento dos discos. Na Figura 4.1 observamos o esquema do arranjo para o espelhamento dos dados. De acordo com a figura temos três discos , disco 0 para o sistema hospedeiro via Linux-VServer, o disco 1 para o primeiro servidor virtual e o disco 2 para o segundo servidor virtual. Ao contrário de propostas anteriores em (PEREIRA, 2005), (ARAUJO, 2006) e (ZAMINHANI, 2008), esta solução apresenta um melhor aproveitamento e planejamento no uso dos recursos computacionais disponíveis em ambos os nós do mini *cluster*.

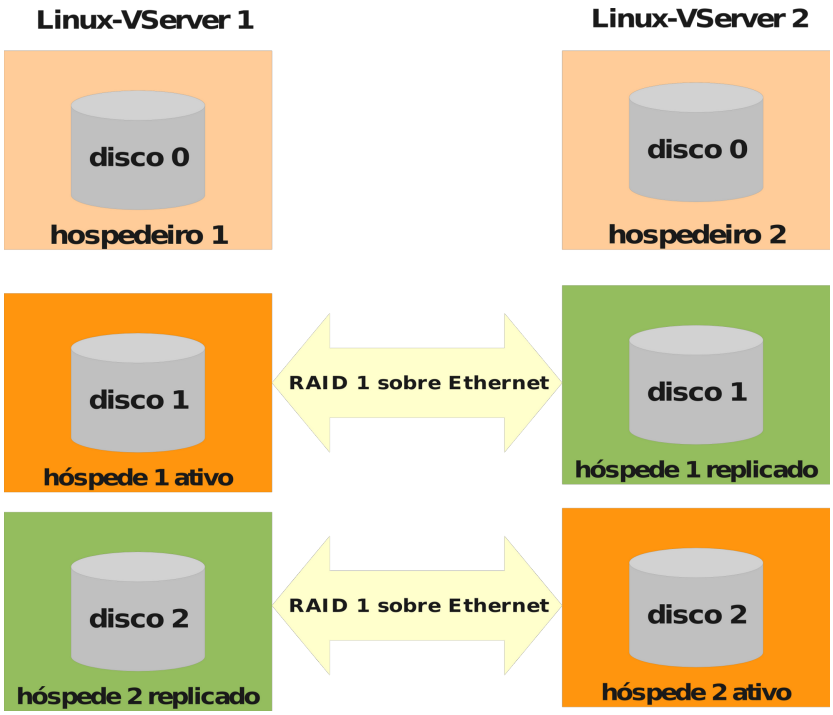

Figura 4.1: Redundância nos serviços e no armazenamento dos dados

A utilização de um disco dedicado ao sistema hospedeiro e um disco para cada servidor virtual objetivou a eliminação de pontos de falha onde a falha de um disco poderia comprometer o sistema como um todo. Cada servidor virtual Linux-

VServer fica responsável por um conjunto de serviços de rede. Cada servidor hospedeiro (instalado no disco 0) mantém ativo um servidor virtual distinto (disco 1 ou disco 2). Os discos 1 e 2 são espelhados via DRBD através de uma interface de rede dedicada. O sistema hospedeiro Linux-VServer é idêntico em ambos os *hosts*, disco 0 em cada servidor físico. Cada servidor físico mantém um servidor virtual ativo e guarda a cópia *on-line* do outro no segundo disco, este fica desativado até que uma eventual falha ocorra e em seguida é iniciado o processo de *failover* de forma transparente aos usuários.

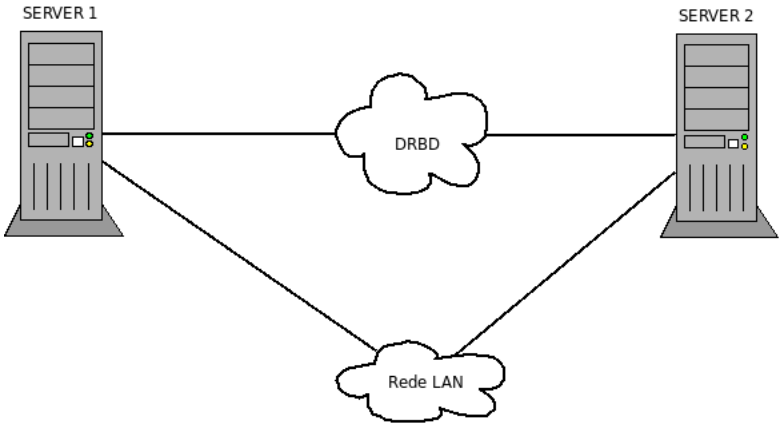

Figura 4.2: Arranjo Lógico

Um segundo ponto chave em um sistema com alta disponibilidade é a capacidade de *failover*, isto é a capacidade de uma máquina, um sistema ou um equipamento de assumir as funções de outro em situações de falha. O *failover* pode ser automático ou manual, sendo o automático o mais desejado e o método empregado nesta proposta. Para implementação do *failover* utilizamos o Heartbeat (descrito no Capítulo 3). O Heartbeat atua tanto como mecanismo de *failover* quanto de monitoramento do sistema e executa o *failover* quando detecta a falha.

Capítulo 5

Implementação

Para implementação da proposta foram utilizados dois servidores com arquitetura AMD x86_64, Dual-Core Opteron com *clock* de 1.8GHz, 4GB de memória RAM, três discos SATA II de 80GB instalados e com duas interfaces de rede. O sistema operacional adotado foi a distribuição GNU/Linux Debian 4.0[1] usando kernel 2.6.18-6-vserver-k7. Os principais *softwares* empregados na implementação foram Heartbeat 2.0.x, DRBD 0.8.0.x e vserver-debiantools 0.3.4.

De forma geral a implementação da proposta seguiu os seguintes passos:

Instalação básica e mínima do sistema operacional Linux Debian com suporte ao *kernel* do Linux-VServer nos dois servidores físicos;

Instalação do módulo DRBD e configuração dos dispositivos DRBD;

Criação e configuração dos contêineres para as máquinas virtuais com o Linux-VServer;

Instalação e configuração do Heartbeat para atuar como o gerenciador do nosso mini *cluster* de alta disponibilidade;

[1] http://www.debian.org

5.1 Instalação do sistema básico e com suporte ao *kernel* do Linux-VServer

A instalação do sistema Linux Debian empregou os recursos da própria distribuição. A distribuição Linux Debian conta com um repositório de pacotes que fornece todos os pacotes necessários. A listagem apresentada na Figura 5.1 ilustra como e quais pacotes foram instalados para ativar o suporte ao Linux-VServer, o k7 indica que em nosso ambiente de implementação foi usado um processador da família AMD. Após a instalação do novo *kernel* foi necessário reiniciar o sistema para que o novo *kernel* fosse carregado. Importante é manter o *kernel* do Linux-VServer como opção padrão no gerenciador de boot.

A instalação foi realizada em ambos os servidores físicos no primeiro disco (disco 0) do nosso arranjo (ver Figura 4.1).

```
# aptitude install linux-image-vserver-k7 vserver-debiantools  \
    util-vserver
```

Figura 5.1: Instalação do *kernel* modificado e as ferramentas para o Linux-VServer.

5.2 Instalação e configuração dos dispositivos DRBD

O DRBD foi instalado usando os pacotes do repositório *backports*[2] do Debian, desta forma é possível trabalhar com uma versão da série 0.8.x do DRBD. Após ativação do repositório *backports* e da atualização da lista de pacotes basta instalar os pacotes drbd8-source e drbd8-utils, algumas dependências também precisam ser instaladas. A listagem exibida na Figura 5.2 mostra a sequência de pacotes instalados e comandos usados para a instalação completa do módulo DRBD.

Uma vez instalado e carregado o módulo do DRBD podemos fazer a configuração dos dispositivos DRBD. A configuração é feita através do arquivo /etc/-drbd.conf, este arquivo deve ser idêntico em ambos os *hosts* com dispositivos DRBD. A listagem apresentada na Figura 5.3 apresenta um exemplo com as principais opções do arquivo de configuração do DRBD.

Na configuração apresentada temos os servidores físicos chamados de "Venus" e "Marte" e cada um deles terá dois dispositivos DRBD, /dev/drbd0 e /dev/-

[2] http://www.backports.org/ - mais detalhes sobre o uso do repositório *backports* podem ser encontrados na sítio oficial.

```
# aptitude install modules-assistant build-essential  \
    kernel-headers
# aptitude -t etch-backports install drbd8-source drbd8-utils
# aptitude install drbdlinks
# m-a a-i drbd8
# depmod -a
# modprobe drbd
```

Figura 5.2: Instalação e ativação do módulo DRBD para o *kernel* do Linux-VServer.

drbd1. Uma vez criado o arquivo de configuração em ambos os servidores devemos efetivamente criar os dispositivos, formatá-los e definir qual será o nó primário. Como teremos dois nós primários ativos, um em cada servidor físico, devemos executar passos semelhantes em ambos os *hosts*. Durante a execução dos comandos foi necessário confirmar algumas informações, bastou seguir as instruções que forem apresentadas na tela. É importante observar que formatamos somente o nó primário, pois a partir da criação do dispositivo e de sua ativação como primário todas as alterações são propagadas para o nó secundário, inclusive a formatação do disco. Nas listagens das figuras 5.4 e 5.5 é mostrada os passos executados em cada *host*, e na listagem apresentada na Figura 5.6 exibe a verificação do estado da sincronização dos discos. Mas detalhes sobre as opções e métodos de configuração dos dispositivos DRBD podem ser obtidos na documentação oficial e nas *man pages* dos comandos.

5.3 Criação dos contêineres para as máquinas virtuais

Primeiramente preparamos o ponto de montagem para os dispositivos DRBD. Os passos serão exemplificados para o servidor físico Venus que manterá ativo o contêiner da máquina virtual **dados1**, mas os mesmos passos foram aplicados no servidor físico Marte que acomodará a máquina virtual **ldap1**[3]. A criação do ponto de montagem pode ser acompanhada na listagem exibida na Figura 5.7, onde criamos o ponto de montagem tanto para o servidor virtual **dados1** quanto para o **ldap1**, pois em momentos de *failover* os dois servidores estarão ativos no mesmo servidor físico. Em seguida é feita a montagem do disco e a criação de um nível extra

[3]Os passos para a criação da máquina virtual no servidor Marte foram os mesmos do servidor venus, respeitando as devidas diferenças e observando a mesma lógica

```
# opcoes globais
global { minor-count 4; }

# os recursos drbd herdam as opcoes abaixo
common {
   # largura de banda usada, opcoes de inicializacao
   syncer { rate  700000; al-extents 257; }
   startup { wfc-timeout  3; degr-wfc-timeout 60; }
   disk { on-io-error detach; } # acao em caso de erro de E/S
}
# configuracao dos dispositivos
resource dsk0 {
   protocol C; # replicacao sincrona
   ( ... )
   on venus { # nome exato retornado pelo comando "uname -n"
      device /dev/drbd0; # nome do dispositivo drbd
      disk /dev/sda1;   # disco ou particao a ser usada
      address 10.1.1.1:7788; # endereco IP e porta usada
      meta-disk internal; # armazena os meta dados no disco
   }
   on marte {
      device /dev/drbd0;
      disk /dev/sda1;
      address 10.1.1.2:7788;
      meta-disk internal;
   }
}
resource dsk1 {
   protocol C;
   ( ... )
   on venus {
      device /dev/drbd1;
      disk /dev/sdb1;
      address 10.1.1.1:7789;
      meta-disk internal;
   }

   on marte {
      device /dev/drbd1;
      disk /dev/sdb1;
      address 10.1.1.2:7789;
      meta-disk internal;
   }
}
```

Figura 5.3: Exemplo de configuração do DRBD e seus respectivos dispositivos.

```
[root@venus ~]# drbdadm create-md all
[root@venus ~]# drbdadm -- --overwrite-data-of-peer primary dsk0
[root@venus ~]# mkfs.ext3 /dev/drbd0
[root@venus ~]# drbdadm primary dsk0
```

Figura 5.4: Criação e ativação como primário e formatação do dispositivo no *host* venus.

```
[root@marte ~]# drbdadm create-md all
[root@marte ~]# drbdadm -- --overwrite-data-of-peer primary dsk1
[root@marte ~]# mkfs.ext3 /dev/drbd1
[root@marte ~]# drbdadm primary dsk1
```

Figura 5.5: Criação e ativação como primário e formatação do dispositivo no *host* Marte.

```
[root@venus ~]# cat /proc/drbd
version: 8.0.11 (api:86/proto:86)
GIT-hash: b3fe2bdfd3b9f7c2f923186883eb9e2a0d3a5b1b build by
    phil@mescal, 2008-02-12 11:56:43
 0: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:3194588 nr:1588624 dw:4860222 dr:749561 al:448 bm:432 lo:0
        pe:0 ua:0 ap:0
    resync: used:0/31 hits:103203 misses:269 starving:0 dirty:0
        changed:269
    act_log: used:0/257 hits:240020 misses:448 starving:0 dirty:0
        changed:448
 1: cs:Connected st:Secondary/Primary ds:UpToDate/UpToDate C r---
    ns:4490 nr:1889039 dw:1893528 dr:42623 al:11 bm:396 lo:0 pe:0
        ua:0 ap:0
    resync: used:0/31 hits:69458 misses:228 starving:0 dirty:0
        changed:228
    act_log: used:0/257 hits:616 misses:11 starving:0 dirty:0
        changed:11
```

Figura 5.6: Verificação de estado atual de sincronização entre os discos DRBD.

de pasta para colocarmos outros arquivos para serem replicados e para garantir a segurança do contêiner[4].

Para criação das máquinas virtuais utilizamos alguns *scripts* fornecidos pelo pacote vserver-debiantools, tais *scripts* facilitam a criação e administração dos contêiners Linux-VServer. O principal comando é o "newvserver" que possui a sintaxe mostrada na listagem da Figura 5.8.

[4] http://linux-vserver.org/Secure_chroot_Barrier

```
[root@venus ~]# mkdir -p /vsrv/{dados1,ldap1}
[root@venus ~]# mount /dev/drbd0 /vsrv/dados1
[root@venus ~]# mkdir -p /vsrv/dados1/barrier
```

Figura 5.7: Criação dos pontos de montagem para o disco DRBD no *host* Venus.

```
# newvserver --vsroot /vservers/ --hostname <nome do host> \
  --domain <dominio>  --ip <endereco IP> --dist etch \
  --mirror <servidor espelho do Debian> --interface <interface de
      rede>
```

Figura 5.8: Sintaxe do comando newvserver.

Descrição dos parâmetros do comando exibido na listagem da Figura 5.8:
vsroot - é o caminho onde os arquivos da máquina virtual ficará. O local padrão é /var/lib/vservers.
hostname - o nome da máquina (por exemplo: mx1)
domain - o nome do domínio do sistema, este nome não precisa ser um domínio real de Internet
ip - endereço IP a ser atribuído para a máquina virtual, pode conter a máscara de sub-rede no formato /24, por exemplo
dist - a distribuição a ser instalada
mirror - o servidor espelho de onde será baixado os arquivos para a instalação
interface - o nome da interface de rede para o endereço IP, por padrão eth0

No arquivo /etc/vservers/newvserver-vars podemos alterar parâmetros padrões de configuração, detalhes na *man page* do comando newvserver. A listagem apresentada na Figura 5.9 ilustra como foi criado o servidor virtual dados1. A criação da máquina virtual poderá levar algum tempo se os arquivos forem baixados diretamente da Internet e dependendo da velocidade de conexão. A máquina virtual será uma instalação mínima e precisará de ajustes e instalação de outros pacotes necessários a implementação dos serviços a serem disponibilizados pelo servidor virtual. Após a criação e configuração básica do primeiro contêiner outros poderão ser criados facilmente através do primeiro, através de cópia. Junto com o contêiner do sistema virtual é criada uma pasta dados1 dentro de /etc/vservers, esta deverá ser copiada para o mesmo local do servidor Marte.

Uma vez criada a máquina virtual podemos inicializá-la e acessá-lá com o comando ”vserver“ que possue a seguinte sintaxe:

```
newvserver --vsroot /vsrv/dados1/barrier/ --hostname dados1 \
   --domain domain.tld  --ip 192.168.80.3/24 --dist etch \
   --mirror http://ftp.br.debian.org/debian/ --interface eth0
```

Figura 5.9: Criação da primeira máquina virtual dados1 no servidor Venus.

vserver <nome no host virtual> [start | stop | restart | enter]

É importante para cada servidor virtual e para cada serviço que estiver ativo no contêiner Linux-VServer ajustar os *daemons* para escutarem somente pelo IP da máquina virtual. Por padrão a maioria dos serviços aguardam requisições globalmente (0.0.0.0) ou somente localmente (127.0.0.0), é preciso configurar cada serviço para que não ocorra erro. Com relação ao *firewall* todo o tráfego deverá ser controlado pela máquina física, ou seja no servidor hospedeiro. É no servidor hospedeiro que as portas do serviços deverão ser liberadas ou bloqueadas.

5.4 Instalação e configuração do Heartbeat

No Debian a instalação do Heartbeat é feita pelo pacote heartbeat-2, este deve ser instalado nos dois servidores físicos e ajustado para inicializar durante o carregamento do sistema. A configuração do Heartbeat se concentra em três arquivos, authkeys, ha.cf e haresources, ambos localizados na pasta /etc/ha.d. Estes arquivos devem ser idênticos em todos os *hosts* físicos do mini *cluster*. O arquivo authkeys é usado para autenticação dos nós do *cluster* e contém informações do tipo e da chave de autenticação. Um exemplo do arquivo authkeys poder ser visto na Figura 5.10.

```
auth 1
1 sha1 43662500f7e40b093cd8120eaa7c2bff71d6e6a1
```

Figura 5.10: Exemplo de arquivo /etc/ha.d/authkeys.

O arquivo ha.cf contém informações importante sobre o *cluster* como a lista de nós, o método e forma de comunicação entre os nós e outras opções de configuração. Um exemplo do arquivo ha.cf utilizado para nossos servidores pode ser visto na listagem da Figura 5.11.

O arquivo haresources contém a configuração dos recursos utilizados pelos nós do *cluster*. Os recursos são especificados um por linha e cada linha é sempre

```
# Heartbeat logging configuration
debugfile  /var/log/ha-debug
logfile  /var/log/ha-log
logfacility local7
use_logd yes

# Heartbeat cluster members
node  venus  ## make sure both names are accessible - check
    /etc/hosts
node  marte

# Heartbeat communication timing
keepalive 2
warntime 10
deadtime 10
initdead 40

# Heartbeat communication paths
bcast eth1
udpport 694

auto_failback on # Auto promotion of primary node upon return to
    cluster.

# Monitoring of network connection to default gateway
ping 192.168.80.1
respawn hacluster /usr/lib/heartbeat/ipfail
```

Figura 5.11: Exemplo de arquivo /etc/ha.d/ha.cf.

lida da esquerda para a direita quando o recurso está sendo ativado, e da direita para a esquerda quando o recurso está sendo desativado e/ou liberado. Um exemplo de arquivo haresources é ilustrado na listagem da Figura 5.12. A dinâmica de processamento do arquivo ocorre da seguinte forma, usando como o exemplo a primeira linha do arquivo (exibido na listagem da Figura 5.12) quando os recursos do nó identificado como ”Venus“ está sendo ativado o Heartbeat promove à primário o dispositivo DRDB (drbddisk::dsk0), monta o dispositivo de acordo com os parâmetros informados (Filesystem::/dev/drbd0::/vsrv/dados1::ext3::defaults) e executa o *script* de inicialização dados1-init passando o parâmetro *start*. O *script* de inicialização se encontra em /etc/init.d/ e fica responsável por levantar ou parar a máquina virtual. Quando o recurso deste mesmo nó esta sendo liberado o Heartbeat executa os mesmo passos mas de maneira inversa e passando um *stop* ao *script* de inicialização, desmontando o disco e rebaixando o dispositivo DRBD à secundário. A listagem exibida na Figura 5.13 exemplifica um arquivo de ini-

cialização de máquina virtual, o *script* inclusive envia uma aviso, por e-mail, ao administrador toda vez que a máquina virtual é iniciada ou parada.

```
venus drbddisk::dsk0  \
  Filesystem::/dev/drbd0::/vsrv/dados1::ext3::defaults \
  dados1-init
marte drbddisk::dsk1 \
  Filesystem::/dev/drbd1::/vsrv/ldap1::ext3::defaults \
  ldap1-init
```

Figura 5.12: Exemplo de arquivo /etc/ha.d/haresources.

```
#!/bin/sh
# Copyright (c) 2008  Wanderson S. Reis <wasare@gmail.com>

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
NAME="dados1"
EMAIL=${EMAIL:-wasare@gmail.com}

test -f `which vserver` || exit 1
test -f `which umount`|| exit 2
test -f `which mail` || exit 3

echo "`hostname -f`" > /tmp/vserver.$$
echo "VServer Guest: $NAME $1" >> /tmp/vserver.$$
. /lib/lsb/init-functions

d_mail() {
    cat /tmp/vserver.$$ | mail -s "[ $NAME $1 ] \
     `date +%d/%m/%Y-%H:%M` "  ${MAIL}
    rm -f /tmp/vserver.$$ 2>&1> /dev/null
}
# Funcao para iniciar a MV
d_start() {
    vserver $NAME start >> /tmp/vserver.$$
    d_mail start
}
# Funcao para parar a MV
d_stop() {
    vserver $NAME stop >> /tmp/vserver.$$
    umount /vsrv/$NAME >> /tmp/vserver.$$
    d_mail stop
}
case "$1" in
    start)
        echo "Starting VServer Guest: $NAME"
        d_start
        ;;
    stop)
        echo "Stopping VServer Guest: $NAME"
        d_stop
        ;;
    *)
        echo "Usage: $0 {start|stop}" >&2
        exit 1
        ;;
esac
exit 0
```

Figura 5.13: Exemplo de *script* de inicialização de máquina virtual.

Capítulo 6

Validação da proposta

Neste capítulo apresentamos a validação da proposta através de simulação de falhas e a verificação da disponibilidade dos serviços durante as falhas.

6.1 Simulação de falhas

O comportamento durante a simulação de falhas foi planejado para mensurar a disponibilidade dos serviços durante uma situação de falha e consequente migração dos serviços de um para outro nó. Para medir a disponibilidade dos serviços considerou-se duas variáveis principais a responsividade da máquina virtual e a responsividade dos serviços específicos. A responsividade da máquina virtual foi aferida através do tempo em segundos em que o nó ficou sem responder ao comando ping. A responsividade dos serviços específicos foram medidas com base na perda de conexões aos serviços durante as falhas simuladas.

As falhas foram simuladas de três formas e com os objetivos descritos a seguir:

Desligamento brusco de uma das máquinas - verificar a capacidade do outro nó de assumir os serviços da máquina desligada em situações de falha elétrica;

Desligamento normal (via *software*) de uma das máquinas - verificar a capacidade do outro nó manter os serviços em situações de manutenção programada ou emergencial;

Desconexão física da rede - verificar a capacidade do outro nó de manter os serviços em situações de falha da placa ou driver da rede.

A listagem da Figura 6.1 exibe o *shell script* utilizado na captura e cálculo do intervalo de indisponibilidade de tempo de resposta ao comando ping.

```
#!/bin/sh
# Copyright (c) 2008  Wanderson S. Reis <wasare@gmail.com>
#
# parametros:
#        endereco IP e numero de sequencia do teste para controle
    (opcional)
#

IP=$1
RETORNO=0

while [ $RETORNO -eq 0 ]
do
   ping -c 1 $1 2> /dev/null 1> /dev/null
   RETORNO=$?
done

INICIAL=`date +%s`

while [ $RETORNO -ne 0 ]
do
   ping -c 1 $1 2> /dev/null 1> /dev/null
   RETORNO=$?
done

FINAL=`date +%s`

INTERVALO=$(($FINAL-$INICIAL))

echo "tempo de transicao no teste $2 host $1 : $INTERVALO s" >
    log/$1-$2.log

exit 0
```

Figura 6.1: *Shell script* que mede o tempo em segundos de indisponibilidade de resposta ao comando ping.

A disponibilidade dos serviços específicos foi medida através de tentativas de conexões aos serviços. Foram cento e cinquenta tentativas de conexão em cada serviço com intervalo de dois segundos para cada tentativa. As conexões foram realizadas no momento em que as falhas eram simuladas. O processo de conexão aos serviços considerou o banco de dados PostgreSQL e o OpenLDAP, cada um

em sua respectiva máquina virtual. A tentativa de conexão partia sempre de uma máquina externa ao conjunto de máquinas virtuais.

As listagens das figuras 6.2 e 6.3 mostram o código PHP utilizado para medir as falhas de conexão ao OpenLDAP e ao PostgreSQL.

Durante a simulação das falhas, as tentativas de conexões aos serviços eram sempre executadas em um ciclo completo de conexão, execução de uma consulta a base OpenLDAP ou a gravação de um número sequencial na base de dados PostgreSQL e a desconexão. As perdas de conexão eram registradas em um arquivo de *log*.

6.2 Disponibilidade dos serviços durante as falhas

A coleta dos dados sobre a disponibilidade foi realizada pelos *scripts* listados nas figuras 6.1 , 6.2 e 6.3. A falha era provocada de forma manual e logo em seguida o *script* para a coleta das informações era executado. Para cada tipo de falha simulada (desligamento brusco, normal e desconexão da rede) e para cada tipo de informação (ping, OpenLDAP e PostgreSQL). Os dados foram coletados três vezes das quais se obteve a média aritmética para cada tipo de informação das combinações tipo falha e tipo de dado. As coletas se limitaram a três pois não foram observadas grandes variações em uma coleta preliminar feita por cinco vezes.

Durante a simulação de falhas foi possível acompanhar todos os passos executados pelo Heartbeat no processo tanto de *failover* quanto de *failback* dos serviços. A listagem da Figura 6.4 ilustra as principais ações executadas pelo Heartbeat durante as falhas.

A Tabela 6.1 apresenta os resultados verificados durante a simulação de falhas. Os momentos mais críticos foram marcados pelo desligamento brusco e pela desconexão física da rede. Nestes momentos ocorreram os maiores intervalos sem resposta do servidor ao comando ping e a maior perda de conexão pelo serviço LDAP. Durante o desligamento normal, para situações programadas, a transição do serviço foi bem rápida e causou o menor impacto na perda de conexão para o LDAP.

De acordo com os dados coletados, nas simulações de falhas, o servidor de banco de dados PostgreSQL apresentou um desempenho excelente com o máximo de duas conexões perdidas independente do cenário. Esta perda representou apro-

Tabela 6.1: Responsividade durante a Simulação de falhas

Falha Simulada	Tempo sem resposta (ping)	Conexões perdidas(*) LDAP / PostgreSQL
Desligamento brusco	40 segundos	6 / 2
Desligamento normal	30 segundos	1 / 2
Desconexão física da rede	35 segundos	5 / 2

* Conexões perdidas em 150 tentativas.

ximadamente 1,33% das conexões realizadas durante os testes. Por sua vez o servidor OpenLDAP se desmonstrou mais frágil durante as transições do serviço de uma máquina para outra alcançando perdas de 4% de conexões no momento das falhas. Considerando o ambiente proposto e os testes é importante observar que a indisponibilidade constatada não comprometeu a continuidade do serviço. Se considerássemos um ambiente de maior porte com mais conexões simultâneas o impacto poderia ser percebido mais facilmente pelos usuários, com perda de algumas transações, contudo ainda seria mantido um bom índice de disponibilidade.

```
<?php
/*
  Copyright (c) 2008  Wanderson S. Reis <wasare@gmail.com>
  Codigo PHP que conecta e consulta uma base LDAP
*/

ini_set('max_execution_time', '1000');
set_time_limit(1000);
require_once 'Benchmark/Timer.php';

define('TENTATIVAS',150);
define('INTERVALO', 2);

$timer = new Benchmark_Timer();
$timer->start();
$timer->setMarker('Mark1');
$timestamp = time();

$f_name = $timestamp .'.log';
$f_pointer = fopen("log/$f_name", 'a+');

for ($i = 1; $i <= TENTATIVAS; $i++)
{
    $ds = @ldap_connect('192.168.10.11');
    ldap_set_option($con, LDAP_OPT_PROTOCOL_VERSION, 3);

    if ($ds) {
        $r = @ldap_bind($ds, 'cn=vserver,ou=Usuario,o=empresa',
            'vserver_password');
        if (!$r)
          fwrite($f_pointer, $f_name .':'. $i .":bind error\n");
    }
    else
        fwrite($f_pointer, $f_name .':'. $i .":connect error\n");

    @ldap_close($ds);
    sleep(INTERVALO);
}

@fclose($f_pointer);
$timer->stop();

echo '<br />'.$timer->timeElapsed();
?>
```

Figura 6.2: Código PHP para consultar uma base OpenLDAP, registrando os erros em um arquivo de *log*.

```
<?php
/*
  Copyright (c) 2008  Wanderson S. Reis <wasare@gmail.com>
  Codigo PHP que conecta a uma base PostgreSQL e
  grava um numero sequencial em uma tabela
*/

ini_set('max_execution_time', '1000');
set_time_limit(1000);
require_once 'Benchmark/Timer.php';

define('TENTATIVAS',150);
define('INTERVALO', 2);

$timer = new Benchmark_Timer();
$timer->start();
$timer->setMarker('Mark1');
$timestamp = time();

$str_conn = 'host=192.168.10.10 port=5432 dbname=vserver
    user=vserver password=vserver';

$f_name = $timestamp .'.log';
$f_pointer = fopen("log/$f_name", 'a+');

for ($i = 1; $i <= TENTATIVAS; $i++)  {
    $conn = @pg_connect($str_conn);
    $result = @pg_query($conn, "INSERT INTO sequencia VALUES
        ($timestamp,$i,". time() .");");
    if  (!$result)
        fwrite($f_pointer, $f_name .':'. $i ."\n");
    @pg_close($conn);
    sleep(INTERVALO);
}

@fclose($f_pointer);
$timer->stop();

echo '<br />'.$timer->timeElapsed();
?>
```

Figura 6.3: Código PHP para gravar em uma tabela do PostgreSQL, registrando os erros em um arquivo de *log*.

```
** Trechos do log do Heartbeat **
venus ipfail: [3163]: info: Status update: Node 192.168.0.1 now
    has status dead
venus heartbeat: [3087]: WARN: node 192.168.0.1: is dead
venus heartbeat: [3087]: info: Link 192.168.0.1:192.168.0.1 dead.
venus harc[8000]: [8003]: info: Running /etc/ha.d/rc.d/status
    status
venus ipfail: [3163]: info: NS: We are dead. :<
venus ipfail: [3163]: info: Link Status update: Link
    192.168.0.1/192.168.0.1 now has status dead
venus ipfail: [3163]: info: We are dead. :<
venus ipfail: [3163]: info: Asking other side for ping node count.
venus ipfail: [3163]: info: Ping node count is balanced.
venus ipfail: [3163]: info: Giving up foreign resources
    (auto_failback).
venus ipfail: [3163]: info: Delayed giveup in 4 seconds.
venus ipfail: [3163]: info: Aborted delayed giveup (6)
venus ipfail: [3163]: info: Link Status update: Link
    192.168.0.1/192.168.0.1 now has status up
venus ipfail: [3163]: info: Status update: Node 192.168.0.1 now
    has status ping

venus kernel: drbd1: peer( Primary -> Secondary )
venus heartbeat: [3087]: info: Received shutdown notice from
    'marte'.
venus heartbeat: [3087]: info: Resources being acquired from
    marte.
venus heartbeat: [11611]: info: acquire local HA resources
    (standby).
venus ResourceManager[11631]: [11643]: info: Acquiring resource
    group: venus drbddisk::dsk0
    Filesystem::/dev/drbd0::/vsrv/dsk0::xfs::defaults dsk0-init
venus heartbeat: [11612]: info: Local Resource acquisition
    completed.
venus Filesystem[11749]: [11755]: INFO: Running status for
    /dev/drbd0 on /vsrv/dsk0
venus Filesystem[11749]: [11759]: INFO: /vsrv/dsk0 is mounted
    (running)
venus Filesystem[11685]: [11760]: INFO: Filesystem Running OK
venus ResourceManager[11631]: [11780]: info: Running
    /etc/init.d/dsk0-init  start
venus heartbeat: [11611]: info: local HA resource acquisition
    completed (standby).
venus heartbeat: [3087]: info: Standby resource acquisition done
    [all].
```

Figura 6.4: Alguns ações executadas pelo Heartbeat durante as falhas simuladas.

Capítulo 7

Conclusão

A proposta e a implementação deste trabalho apresentou uma solução simples e viável de alta disponibilidade com uso de virtualização baseada em contêineres GNU/Linux. Na aplicação da solução em um ambiente real foi possível demonstrar a capacidade de se manter a disponibilidade do sistema mesmo diante de situações de falha. O tempo de transição dos serviços de uma máquina virtual para outra foi bastante satisfatória, ficando bem abaixo de um minuto e as perdas de comunicação durante este período foram aceitáveis.

Ao contrário de outras soluções semelhantes existe um melhor aproveitamento do *hardware* empregado, pois apesar da redundância ela não fica completamente ociosa visto que existem dois sistemas virtuais com serviços ativos em cada nó. Não houve a necessidade de uso de *hardware* especial. O administrador do sistema poderá realizar manutenções e atualização de *harware* sem uso de janela de tempo. Outros requisitos importantes alcançados na estruturação e implementação da proposta foram o emprego de tecnologias de *software* livre com todos os seus benefícios agregados como menor custo de implantação, boa relação custo benefício de manutenção, facilidade de redimensionamento e escalabilidade de recursos sem aquisição de licenças especiais e o uso de tecnologia inovadora.

Como proposta para trabalhos futuros seria interessante o acréscimo de mais nós respondendo por outros serviços e ficando com uma redundância dupla, para o caso de três nós. Outra possibilidade seria a estruturação da mesma proposta com o emprego de outras tecnologias de virtualização como o Xen ou KVM, para efeitos comparativos.

Referências Bibliográficas

ARAUJO, E. S. Monografia (Bacharelado em Sistemas de Informação), *Implementando sistemas de alta disponibilidade de serviços utilizando software livre*. Imperatriz - MA: [s.n.], 2006.

Ellenberg, L. DRBD 8.0.x and beyond Shared-Disk semantics on a Shared-Nothing Cluster. 10 ago. 2007. Disponível em: <http://www.drbd.org-/fileadmin/drbd/publications/drbd8.linux-conf.eu.2007.pdf>. Acesso em: jan. 2009.

GOVERNO BRASILEIRO. *Guia de Estruturação e Administração do Ambiente de Cluster e Grid*. 1.0. ed. Brasília, 1 abr. 2007. Disponível em: <https://www.governoeletronico.gov.br/anexos/guia-de-cluster>. Acesso em: abr. 2008.

HAAS, F.; REISNER, P.; ELLENBERG, L. *The DRBD User's Guide*. [S.l.]. Disponível em: <http://www.drbd.org/users-guide/>. Acesso em: fev. 2009.

IBM. *IBM Systems: Virtualization*. Version 2 release 1. [S.l.], dez. 2005. Disponível em: <http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic-/eicay/eicay.pdf>. Acesso em: mar. 2009.

IKE, F. Alta disponibilidade. *Linux Magazine*, n. 43, p. 31, jun. 2008.

JONES, M. T. Virtual Linux - An overview of virtualization methods, architectures, and implementations. *IBM - developerworks*, 29 dez. 2006. Disponível em: <http://www.ibm.com/developerworks/library% -/l-linuxvirt/>. Acesso em: mar. 2008.

LINUX-HA PROJECT. *About the Linux High Availability Project*. [S.l.], 2007. Disponível em: <http://www.linux-ha.org/>. Acesso em: jan. 2008.

MATTOS, D. M. F. Virtualização: VMWare e Xen. 2008. Disponível em: <http://www.gta.ufrj.br/grad/08_1/virtual/artigo.pdf>. Acesso em: mar. 2009.

NEVES, G.; PACHECO, G. Xen Hypervisor. nov. 2008. Disponível em: <http://www.lisha.ufsc.br/~guto/teaching/os/ine5412-2008-2/work/xen.pdf>. Acesso em: mar. 2009.

OLIVEIRA, G. V. N. Monografia (Especialização em Administração em Redes Linux), *Solução de virtualização completa utilizando VMware e Software Livre: Um Estudo de Caso na CEF*. Lavras - MG: [s.n.], abr. 2007. Disponível em: <http://www.ginux.ufla.br/node/168>. Acesso em: mar. 2009.

PADALA, P.; ZHU, X.; WANG, Z.; SINGHAL, S.; SHIN, K. G. Performance Evaluation of Virtualization Technologies for Server Consolidation. *Enterprise Systems and Software Laboratory*, HP Laboratories Palo Alto, n. HPL-2007-59, nov. 2007. Disponível em: <http://www.hpl.hp.com/techreports/2007/HPL-2007-59.pdf>. Acesso em: mar. 2008.

PEREIRA, R. B. O. Monografia (Especialização em Administração em Redes Linux), *Alta disponibilidade em sistemas GNU/LINUX utilizando as ferramentas DRBD, Heartbeat e Mon*. Lavras - MG: [s.n.], 12 set. 2005. Disponível em: <http://www.ginux.ufla.br/node/120>. Acesso em: mar. 2008.

PIEDAD, F.; HAWKINS, M. *High Availability: Design, Techniques, and Processes*. [S.l.]: Prentice Hall, 2000.

SILVA, R. F. Monografia (Graduação em Tecnologia da Informação e da Comunicação), *Virtualização de sistemas operacionais*. Petrópolis - RJ: [s.n.], 2007. Disponível em: <http://www.lncc.br/~borges/doc-/Virtualiza%e7%e3o%20de%20Sistemas%20Operacionais.pdf>. Acesso em: mar. 2009.

SOLTESZ, S.; POTZL, H.; FIUCZYNSKI, M. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: *Proceedings of EuroSys 2007*. Lisbon, Portugal: [s.n.], 2007. Disponível em: <http://www.cs.princeton.edu/~mef/research/vserver/paper.pdf>. Acesso em: mar. 2008.

WEBER, T. S. *Um roteiro para exploração dos conceitos básicos de tolerância a falhas*. 2002. Disponível em: <http://www.inf.ufrgs.br/~taisy/disciplinas/textos-/Dependabilidade.pdf>. Acesso em: mar. 2009.

ZAMINHANI, D. Monografia (Especialização em Administração em Redes Linux), *Cluster de alta disponibilidade através de espelhamento de dados em máquinas remotas*. Lavras - MG: [s.n.], abr. 2008. Disponível em: <http://www.ginux.ufla.br/node/236>. Acesso em: mar. 2009.