

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RN
ESPECIALIZAÇÃO EM REDES DE COMPUTADORES**

BELY SILVA JUNIOR

**IMPLEMENTAÇÃO DE SOLUÇÃO DE ALTA DISPONIBILIDADE PARA PROXY
HTTP UTILIZANDO FERRAMENTAS LIVRE**

**NATAL/RN
2012**

BELY SILVA JUNIOR

**IMPLEMENTAÇÃO DE SOLUÇÃO DE ALTA DISPONIBILIDADE PARA PROXY
HTTP UTILIZANDO FERRAMENTAS LIVRE**

Monografia apresentada ao curso de Especialização em Redes de Computadores do Centro Universitário do Rio Grande do Norte como requisito final para a obtenção do título de Especialista em Redes de Computadores.

Orientador: Prof. M.Sc. Aluízio
Ferreira da Rocha Neto

Co-Orientador: Prof. M.Sc. Francisco
Sales de Lima Filho

**NATAL/RN
2012**

BELY SILVA JUNIOR

**IMPLEMENTAÇÃO DE SOLUÇÃO DE ALTA DISPONIBILIDADE PARA PROXY
HTTP UTILIZANDO FERRAMENTAS LIVRE**

Monografia apresentada ao curso de Especialização em Redes de Computadores do Centro Universitário do RN - UNI-RN como requisito final para obtenção do título de Especialista em Redes de Computadores.

Aprovado em: ____/____/____

BANCA EXAMINADORA

Prof. Nome Completo Orientador

Membro Examinador

Membro Examinador

AGRADECIMENTOS

Gostaria de agradecer a Ana Luisa, minha noiva, por ter sempre me apoiado para conclusão deste trabalho, a minha família por me fazer chegar ate aqui e ao grande M.Sc. Francisco Sales de Lima Filho pela grande ajuda no desenvolvimento deste trabalho, tanto com a disponibilização de ambiente, sempre ajudando com suas grandes ideias e motivação para desenvolvimento deste.

RESUMO

A alta disponibilidade em ambientes onde a tecnologia da informação é inserida como parte da instituição, onde geralmente todos os serviços estratégicos estão encaminhados e em alguns casos totalmente informatizados, como é o caso dos sistemas de recursos humanos, e-mail, armazenamento de arquivos entre outros sistemas, é de fato imprescindível o uso de tecnologias voltadas a disponibilizar clusters de alta disponibilidade e desempenho. Este trabalho objetiva descrever a implantação de um ambiente com função de *Proxy-cache* em alta disponibilidade, com utilização de ferramentas *Open Sources*, onde será demonstrado a implantação de ferramentas como *corosync* e *pacemaker* para implementação de alta disponibilidade e criação de ambiente em *LVS (LINUX VIRTUAL SERVER)* objetivando o alto desempenho de ambiente com função de *Proxy Cache* com a utilização da ferramenta *Squid*.

Palavras-Chaves: serviços críticos, alta disponibilidade, software livre, Proxy, squid

ABSTRACT

High availability in environments where information technology is included as part of the institution, where usually all strategic services are directed and in some cases fully computerized, as is the case of human resource systems, e-mail, file storage between other systems, it is indeed essential to the use of technologies for disponibilizar clusters for high availability and performance. This paper aims to describe the implementation of an environment with role-Proxy cache for high availability, using tools Open Sources, which will be demonstrated deployment tools like corosync and pacemaker for implementing high availability and creation of environment for LVS (Linux VIRTUAL SERVER) aiming high performance environment with function Proxy Cache using Squid tool.

Key Words: critical services, high availability, software free, Proxy, squid

LISTA DE FIGURAS

Figura 1 - Topologia de ambiente com Proxy.....	14
Figura 2 - Topologia Balanceamento de carga.....	15
Figura 3 - Topologia de <i>cluster</i> LVS configurado como NAT.	18
Figura 4 - Topologia de <i>cluster</i> LVS configurado como Túnel.	19
Figura 5 - Topologia de <i>cluster</i> LVS configurado como Roteamento direto.....	19
Figura 6 - Topologia com uso corosync.....	20
Figura 7 - Topologia completa da solução, com roteamento direto, contemplando balanceamento de carga e tolerância a falhas.....	24
Figura 8 - Balanceamento de carga de 100 (cem) conexões, utilizando algoritmo <i>Round Robin</i> , para os servidores de aplicação do cluster.	25
Figura 9 - Balanceamento de carga de 100 (cem) conexões, com falha em um servidor de aplicação do <i>cluster</i>	26
Figura 10 - Recuperação de falha em nó de aplicação.....	26
Figura 11 – Chaveamento automático do lvs01 para o lvs02, evitando que o <i>cluster</i> ficasse indisponível.....	27

SUMÁRIO

1	INTRODUÇÃO.....	9
1.1	PROBLEMA DE PESQUISA	9
1.2	OBJETIVOS	10
1.2.1	Objetivos Gerais	10
1.2.2	Objetivos Especificos.....	10
2	REFERENCIAL TEORICO	11
2.1	SOFTWARE LIVRE.....	11
2.2	SISTEMA OPERACIONAL GNU/LINUX	12
2.3	SERVIDOR DE PROXY SQUID	13
2.4	ALTA DISPONIBILIDADE EM SERVIÇOS	14
2.4.1	Serviços com balanceamento de carga	15
2.4.2	Serviços com tolerância a falhas	16
2.5	FERRAMENTAS LIVRES PARA IMPLANTAÇÃO DE ALTA DISPONIBILIDADE.....	17
2.5.1	LVS (Linux Virtual Server).....	17
2.5.2	COROSYNC	20
2.5.3	PACEMAKER	20
2.5.4	CSYNC2	21
3	IMPLEMENTAÇÃO DE SOLUÇÃO DE ALTA DISPONIBILIDADE PARA PROXY HTTP UTILIZANDO FERRAMENTAS LIVRE	23
3.1	METODOLOGIA ADOTADA.....	23
3.2	TOPOLOGIA DA SOLUÇÃO.....	23
3.3	RESULTADOS.....	24
3.3.1	Primeiro Experimento - Balanceamento De Carga.....	25
3.3.2	Segundo Experimento – Tolerância a Falha em (N-1) Nós de Aplicação.....	26
3.3.3	Terceiro Experimento – Recuperação de Falha em Nó de Aplicação	26
3.3.4	Quarto Experimento – Tolerância a Falha em Nó Balanceador de Carga.....	27
4	CONSIDERAÇÕES FINAIS.....	28
4.1	CONTRIBUIÇÕES DO TRABALHO	28
4.2	TRABALHOS FUTUROS	29
	REFERÊNCIAS BIBLIOGRÁFICAS	30
	ANEXO I - TUTORIAL DE INSTALAÇÃO E CONFIGURAÇÃO.....	31

1 INTRODUÇÃO

Os sistemas de informações vêm se tornando cada vez mais estratégico para empresas e instituições públicas, seja para sustentar processos de rotinas internas, ou mesmo para garantir a competitividade sustentando processos estratégicos para o negócio da companhia. São exemplos de rotinas e processos internos comuns a todas as organizações a gestão de recursos humanos e gestão de recursos financeiros. Dependendo da área de atuação, o sítio das organizações, o acesso à internet e as lojas virtuais são considerados processos estratégicos para o negócio.

Em razão disso, a área de Tecnologia da Informação – TI vem demonstrando que é um setor de grande importância e deve ser tratada como setor estratégico para o negócio da instituição.

Com todo esse crescimento, manter os sistemas críticos para o negócio disponíveis e com desempenho adequado tornou-se imperativo, abrindo frentes de pesquisa para desenvolver soluções de sistemas redundantes, eliminando pontos únicos de falhas.

Neste contexto, fabricantes de grande porte como a *Microsoft®* desenvolveram produtos especializados para prover tolerância a falhas, como o *Microsoft Cluster Service*. Entretanto, tais produtos possuem valor aquisitivo bastante alto, tornando-se inviável para empresas com orçamento reduzido, obrigando-as a buscar soluções alternativas de menor custo.

Neste cenário, foram desenvolvidas ferramentas de código livre que possibilitar criar ambientes redundantes, de alto desempenho adequado tanto para pequenas como para grandes empresas.

Tais ferramentas foram utilizadas neste trabalho para implementar uma solução de alta disponibilidade para serviço de acesso à internet utilizando ferramentas livres.

1.1 PROBLEMA DE PESQUISA

Nos últimos anos, o crescimento do uso da Internet possibilitou que empresas de todos os setores compartilhassem informações, que lojas virtuais se consolidassem, a expansão das redes sociais dentre outros serviços disponíveis na

grande rede. Esse movimento fez crescer também a necessidade de manter o serviço de acesso à internet das empresas tolerante à falhas e com o nível de segurança adequado às suas políticas internas.

Para tanto, soluções abertas que utilizem ferramentas para controle de acesso, filtro e controle de conteúdo, que garantam alta disponibilidade do serviço de acesso à internet são fundamentais nas instituições de hoje que não pretendem investir em soluções proprietárias e caras.

Por este motivo, uma solução de Proxy de acesso à internet, em ambiente redundante, com o nível de desempenho adequado, que permita ao usuário acessar informações de forma rápida e segura revela-se como tema importante para pesquisa. .

1.2 OBJETIVOS

1.2.1 Objetivos Gerais

Esse trabalho tem como finalidade compilar informações relativas à soluções de alto desempenho e tolerância à falhas, além de implementar uma solução de alta disponibilidade para *proxy* http utilizando ferramentas livre, demonstrando, de forma pratica e clara, a configuração das ferramentas envolvidas.

1.2.2 Objetivos Especificos

Para atingir os objetivos gerais, foram realizadas as seguintes atividades:

- Pesquisa, compilação, instalação e testes dos *softwares* envolvidos;
- Implementação, teste e análise do ambiente de alta disponibilidade;
- Documentação do ambiente, testes e resultados.

2 REFERENCIAL TEORICO

Neste capítulo serão descritos os aspectos teóricos necessários ao entendimento do trabalho como um todo.

2.1 SOFTWARE LIVRE

Em 1984, Richard Stallman, funcionário do Laboratório de Inteligência Artificial do MIT¹, passou por uma experiência negativa com *software* comercial e deu origem ao projeto GNU². Durante o período que estava no MIT identificou uma falha no *software* de uma impressora Xerox. Tentou corrigi-la, mas a empresa não liberou o código-fonte. Esse fato motivou Richard Stallman a criar um mecanismo legal de garantia para que todos pudessem desfrutar dos direitos de copiar, redistribuir e modificar *software*, dando origem a Licença GPL³. Para institucionalizar o projeto GNU, Stallman fundou a *Free Software Foundation*- FSF, nasce assim o movimento do *Software* Livre (Wikipédia).

A licença GPL idealizada por Richard Stallman é baseada nas liberdades que a FSF defende, são elas:

- Liberdade de executar o programa, para qualquer propósito (liberdade zero);
- Liberdade de estudar como o programa funciona e adaptá-lo às suas necessidades (liberdade 1), sendo o acesso ao código-fonte um pré-requisito para esta aspecto;
- Liberdade de distribuir cópias de forma que você possa ajudar ao seu próximo (liberdade 2);
- Liberdade de melhorar o programa e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie (liberdade 3). Novamente, aqui o acesso ao código-fonte é um pré-requisito.

¹ MIT – O Instituto de Tecnologia de Massachusetts (*Massachusetts Institute of Technology*) é um centro universitário de educação e pesquisa privado localizado em Cambridge, Massachusetts, nos Estados Unidos.

² Projeto GNU – O Projeto GNU foi concebido em 1983 como uma maneira de trazer de volta o espírito cooperativo que prevalecia na comunidade de computação nos seus primórdios para tornar a cooperação possível novamente ao remover os obstáculos à cooperação impostos pelos donos de software proprietário.

³ Licença GPL – A Licença Pública Geral do GNU é frequentemente chamada abreviadamente de GNU GPL e é utilizada pela maioria dos programas do GNU assim como muitos outros programas de software livre que não são parte do Projeto GNU.

Para inserir um *software* na GPL, ele deve conter, no mínimo, as três liberdades citadas anteriormente.

O GNU/Linux, idealizado por Linus Torvalds, é atualmente considerado o principal projeto do *software* livre, estando presente em estações de trabalho e em servidores de empresas de pequeno, médio e grande porte.

2.2 SISTEMA OPERACIONAL GNU/LINUX

O sistema operacional GNU/Linux é uma versão de livre distribuição do Unix, originalmente desenvolvido por Linus Torvalds, que começou a trabalhar no Linux em 1991, como estudante da Universidade de Helsinki, na Finlândia. (DALHEIMER, DAWSON, *et al.*, 2002).

Baseado no Minix⁴, idealizado por Andrews Tanenbaum, o Linux absorveu algumas características UNIX⁵, como a capacidade de multitarefa, multiprocessamento, uso de encadeamentos no kernel (*kernel threading*), preempção do *kernel*, suporte a aplicações multi-encadeadas (*multithreaded application support*) e sistema de arquivos baseado em VFS (*Virtual File System*).

O kernel Linux é o sistema operacional, ele é o responsável por todo controle de periféricos e dispositivos do sistema.

Silva (2002) descreve:

“O kernel faz o controle dos periféricos do sistema e para isto ele deve ter o seu suporte incluído. Para fazer uma placa de som *Sound Blaster* funcionar, por exemplo, é necessário que o kernel ofereça suporte a esta placa e você deve configurar seus parâmetros (como interrupção, I/O e DMA) com comandos específicos para ativar a placa e fazê-la funcionar corretamente. Existe um documento que contém quais são os periféricos suportados/ não suportados pelo GNU/Linux, ele se chama Hardware-HOWTO.

Suas versões são identificadas por números como 2.0.36, 2.0.38, 2.1.10, 2.2.12, as versões que contém um número par entre o primeiro e segundo ponto são versões estáveis e que contém números ímpares neste mesmo local são versões instáveis (em desenvolvimento). Usar versões instáveis não quer dizer que ocorrerá travamentos ou coisas do tipo, mas algumas

⁴ Minix – O Minix é um sistema operacional Unix-like (semelhante ao UNIX), escrito em C (linguagem de programação) e assembly

⁵ UNIX - Unix é um sistema operativo (ou sistema operacional) portátil (ou portátil), multitarefa e multiutilizador (ou multiusuário) originalmente criado por Ken Thompson, Dennis Ritchie, Douglas McIlroy e Peter Weiner, que trabalhavam nos Laboratórios Bell (Bell Labs) da AT&T.

partes do kernel podem não estar testadas o suficiente ou alguns controladores podem ainda estar incompletos para obter pleno funcionamento. Se opera sua máquina em um ambiente crítico, prefira pegar versões estáveis do kernel.”

Hoje, o Linux é de autoria e mantido por milhares de desenvolvedores que colaboram livremente através da Internet. Empresas têm surgido para fornecer suporte ao Linux, para empacotá-lo em distribuições e vender estações de trabalho pré-instalados com o *software* Linux (DALHEIMER, DAWSON , *et al.*, 2002).

Dentre as distribuições Linux, as mais conhecidas são:

- *Debian GNU/Linux*, disponível em <http://www.debian.org/>;
- *Slackware Linux*, disponível em [http://www.slackware.com](http://www.slackware.com;);
- *Red Hat Linux*, disponível em [http://www.redhat.com](http://www.redhat.com;);
- *Ubuntu Linux*, disponível em <http://www.ubuntu.com>.

Existe uma enorme quantidade de distribuições Linux, algumas de uso geral, mais utilizadas por usuários domésticos, outras mais específicas, utilizadas por empresas e, devido a flexibilidade do sistema, existem distribuições altamente especializadas em várias áreas do conhecimento. Essa versatilidade do Linux tem possibilitado sua consolidação e utilização em todo o mundo.

Com consolidação do Linux, aliada a grande adesão à filosofia do *software* livre, uma enorme quantidade de ferramentas vem sendo desenvolvidas e aperfeiçoadas por diversos grupos e disponibilizados na internet. Todas as ferramentas utilizadas neste trabalho foram obtidas livremente na internet.

2.3 SERVIDOR DE PROXY SQUID

O servidor Proxy⁶ Squid foi desenvolvido da década de 90, baseado no sistema *Harvest Cache Daemon*⁷. Hoje, especializado nos protocolos HTTP⁸,HTTPS⁹ e

⁶ Proxy – Programa que funciona como intermediário entre o usuário final e o servidor, atendendo a requisições e repassando.

⁷ Harvest Cache Daemon E é um HTTP/1.0 proxy completo. Desenvolvido na década de 90.

⁸ HTTP – O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web.

⁹ HTTPS - HyperText Transfer Protocol Secure, é uma implementação do protocolo HTTP sobre uma camada adicional de segurança que utiliza o protocolo SSL/TLS.

FTP¹⁰, o projeto é mantido por colaboradores do mundo inteiro, sendo considerado por profissionais da área de TI como uma ferramenta robusta, versátil e confiável.

Atualmente, o squid-cache é uma das ferramentas *open sources*¹¹ mais utilizadas para acelerar a navegação na internet, em razão da boa performance de gerenciamento objetos em *cache*, bem como para implementar o controle de acesso através de filtros de conteúdo.

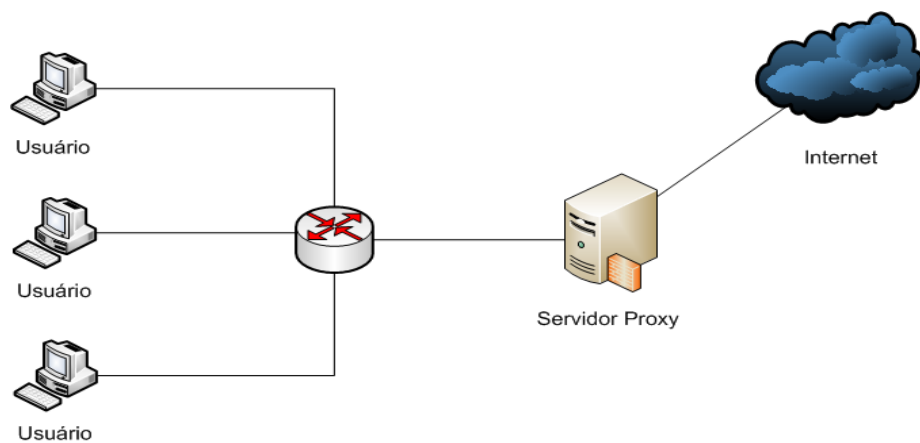


Figura 1 - Topologia de ambiente com Proxy.

O trabalho do *Squid* é ser um intermediário em uma transação web, como pode ser observado na Figura 1. Ele aceita um pedido de um cliente, processa o pedido e encaminha a solicitação para o servidor de origem. O pedido pode ser registrado, rejeitado, e até mesmo modificado antes de encaminhar. Como um cache, *Squid* armazena os recentes conteúdos web para possível reutilização posterior. As solicitações subsequentes para o mesmo conteúdo pode ser servido a partir do cache, ao invés de entrar em contato com o servidor de origem novamente (WESSELS, 2004).

2.4 ALTA DISPONIBILIDADE EM SERVIÇOS

De forma geral, a alta disponibilidade de serviços pode ser definida em função do tempo que um sistema permanece útil ao propósito para o qual foi

¹⁰ FTP –Protocolo para transferência de arquivos (File Transfer Protocol).

¹¹ *Open Sources* – Foi criada pela *Open Source Initiative* (OSI) a partir do texto original da *Debian Free Software Guidelines* (DFSG) e Genericamente trata-se de software que respeita as quatro liberdades definidas pela *Free Software Foundation*.

desenvolvido. Entretanto, para que os serviços operem normalmente, é necessário que os vários componentes da solução não falhem. Com isso, surge a importância de implantar redundância nos componentes da solução, evitando a existência de pontos de falhas. Quanto menores forem os SPOF(*Single Point Of Failure*), e menor será a probabilidade de interrupções no serviço.

Normalmente, o passo que antecede a implementação de sistemas de alta disponibilidade é o mecanismo de balanceamento de carga que juntos formam um sistema de alto desempenho com tolerância a falhas. O balanceamento de carga é descrito na seção 2.4.1 e serviços tolerantes a falhas na sessão 2.4.2.

2.4.1 Serviços com balanceamento de carga

O balanceamento de carga tem como objetivo aumentar a capacidade do serviço através da distribuição das requisições entre os nós pertencentes ao arranjo de balanceamento.

A Figura 2 apresenta uma topologia clássica de balanceamento, onde as requisições são recebidas pelo nó balanceador e distribuídas para os nós de aplicação.

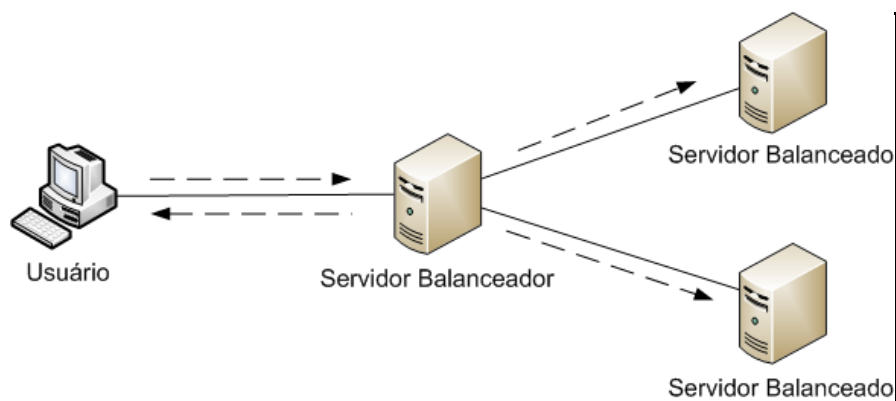


Figura 2 - Topologia Balanceamento de carga.

Para realizar o balanceamento, o nó balanceador utiliza um algoritmo de balanceamento. Existem vários desses algoritmos, merecendo destaque os algoritmos *Round Robin*, *Weighted Round-Robin*, *Least Connection* e *Weighted Least* detalhados abaixo:

- **Round Robin:** é um dos algoritmos mais simples de agendamento de processos em um sistema operacional, que atribui frações de tempo para cada processo em partes iguais e de forma circular, sem manipulação de todos os processos prioridade. Escalonamento *Round-Robin* é simples e fácil de implementar (Wikipedia, 2012);
- **Weighted Round-Robin:** o agendamento de repetição alternado é a escolha preferida caso existam diferenças significantes na capacidade de servidores reais no *pool*. No entanto, se uma solicitação de carga variar dinamicamente, o servidor de carga mais pesada talvez responda mais do que suas solicitações compartilhadas (REDHAT, 2012);
- **Least Connection:** distribui mais solicitações aos servidores reais com poucas conexões ativas. A conexão mínima é um tipo de algoritmo de agendamento dinâmico, fazendo com que isto seja a melhor opção caso haja um alto grau de variações na carga solicitada (REDHAT, 2012)
- **Weighted Least:** distribui mais pedidos para os servidores com menos conexões ativas em relação às suas capacidades (REDHAT, 2012);

2.4.2 Serviços com tolerância a falhas

Serviços com tolerância a falhas são implementados através de mecanismos de redundância, eliminando pontos únicos de falhas. Uma forma comum de implementar esse tipo de tolerância é através de *clusters*. Entende-se por *cluster* a união de dois ou mais computadores trabalhando em conjunto com o objetivo de aumentar a capacidade de processamento, ou aumentar a resiliência¹² do serviço.

É muito comum encontrar ambientes em *cluster* em instituições que necessitam de um nível alto de processamento ou de alta disponibilidade, como provedores de serviços de Internet, lojas virtuais e *Internet Banking*, dentre outros.

Os principais tipos de *clusters* agrupam recursos de alta disponibilidade e balanceamento de carga. A combinação dessas duas características possibilitam a criação de ambientes robustos e com grande grau de tolerância a falha.

¹² Resiliência – Elasticidade ou capacidade do sistema se recuperar de uma falha.

Ambientes de alta disponibilidade também devem permitir o chaveamento para o componente reserva quando o principal falhar (*Failover*), bem como o retorno à situação normal quando o equipamento principal retornar a normalidade (*Failback*). Essas características são melhor detalhadas abaixo:

- **Failover:** em ambientes redundantes, com servidores equivalentes, é responsável por uma máquina assumir as funções da outra em momentos de detecção de falha no servidor principal. Assim disponibilizando o servidor defeituoso para manutenção. O processo pode ser automático ou manual, isso vai variar de acordo com a necessidade e política de cada organização.
- **Failback:** este processo é o inverso ao *Failover*, após a detecção a falha e desativação do servidor principal. É natural que este serviço seja levado à manutenção e realizado as devidas correções, após normalização do servidor o servidor é novamente colocado em produção e automaticamente assumirá o posto de servidor primário, esta é a função do *Failback*.

Na seção seguinte são apresentadas as ferramentas que foram utilizadas para implementar a solução de alta disponibilidade para *Proxy HTTP* utilizando ferramentas livre.

2.5 FERRAMENTAS LIVRES PARA IMPLANTAÇÃO DE ALTA DISPONIBILIDADE

Nesta seção serão apresentadas as ferramentas utilizadas para implementar a solução de alta disponibilidade utilizando ferramentas livre.

2.5.1 LVS (Linux Virtual Server)

O Linux Virtual Server - LVS é um projeto *Open Source* que busca proporcionar ambientes com alto nível de desempenho e disponibilidade, objetivando manter uma estrutura escalável e confiável.

O projeto LVS utiliza recursos de outro projeto, o *IP Virtual Server* – IPVS, implementado diretamente no kernel do Linux, onde tem a função de *switch* simula endereços IP virtual para ser utilizado em conjunto com *clusters*.. O LVS pode ser configurado de três maneiras distintas, conforme descrito abaixo:

- **Configurado como NAT¹³:** a estrutura do *cluster* possui apenas um endereço público no SERVIDOR DIRETOR¹⁴, os demais servidores reais serão máquinas com endereços IP privados, roteados pelo módulo *ip_foward* encontrado no Kernel Linux. O fato de possuir apenas um endereço público faz com que todas as solicitações sejam direcionadas e respondidas pelo servidor responsável pelo balanceamento, ou seja, o SERVIDOR DIRETOR. A principal vantagem dessa abordagem é não existir limitação quanto ao sistema operacional, bastando apenas que a pilha de protocolo TCP/IP seja suportada. A Figura 3 ilustra a utilização do LVS configurado com NAT.

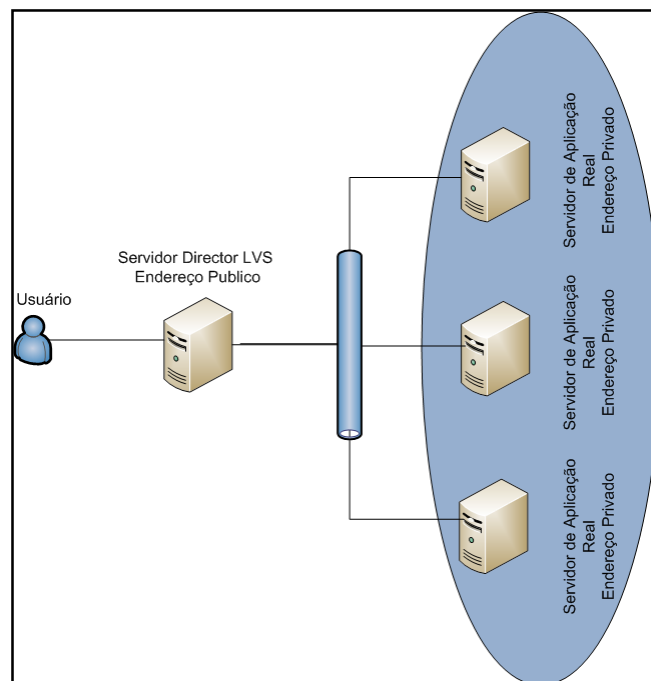


Figura 3 - Topologia de *cluster* LVS configurado como NAT.

- **Configurado como Túnel:** neste modelo, os servidores devem suportar a implementação de *IP Tunneling*, pois as requisições chegarão ao servidor

¹³ NAT – Network Address Translator, mecanismo utilizado para fazer tradução e mapeamento de endereços de rede.

¹⁴ SERVIDOR DIRETOR- Servidor responsável pela gerência do *cluster*.

SERVIDOR DIRETOR, serão encaminhadas aos servidores de aplicação que responderão diretamente para o cliente solicitante, conforme ilustrado na Figura 4.

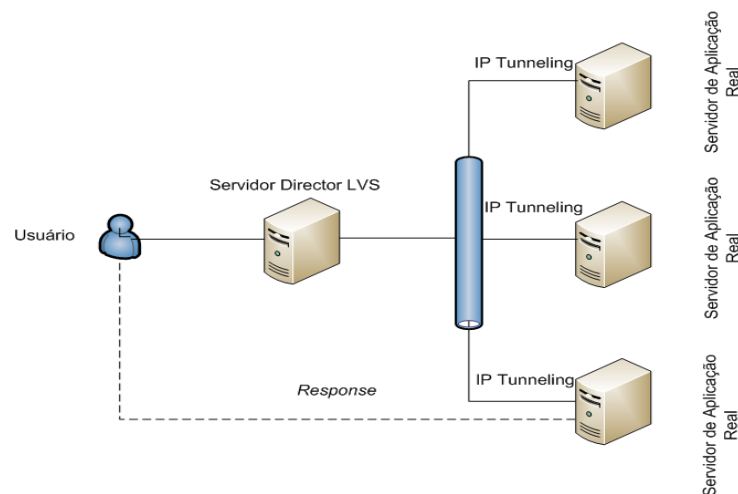


Figura 4 - Topologia de *cluster* LVS configurado como Túnel.

- **Configurado como Roteamento Direto:** neste modelo, o SERVIDOR DIRETOR é configurado com endereço público e os servidores de aplicação, através de *aliases*, também com o mesmo endereço público. Como no modelo de Túnel, as requisições serão encaminhadas pelo SERVIDOR DIRETOR aos servidores de aplicação que responderão diretamente para o cliente solicitante, conforme ilustrado na Figura 5.

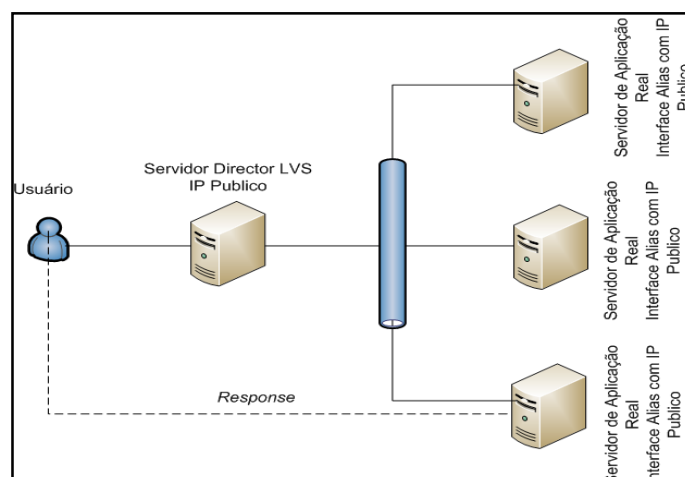


Figura 5 - Topologia de *cluster* LVS configurado como Roteamento direto.

O LVS mostraram-se, juntamente com as ferramentas que serão apresentadas nas seções a seguir, uma alternativa atraente para implementar ambientes de alta disponibilidade.

2.5.2 COROSYNC

O *corosync* é uma ferramenta *Open Source* desenvolvida para auxiliar ambientes de alta disponibilidade, tem como função principal monitorar os nós do *cluster*, verificando suas respectivas disponibilidades.

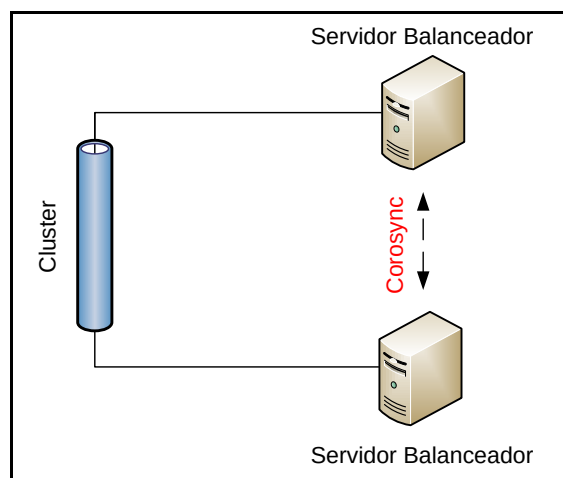


Figura 6 - Topologia com uso corosync.

O *corosync* como demonstrado na Figura 6 é ferramenta importante para implementação de ambiente de alta disponibilidade, monitorando os nós do cluster.

2.5.3 PACEMAKER

O *pacemaker* é um gerenciador de *cluster*, responsável por manter os serviços o máximo disponível, com recursos de detecção e recuperação de falhas em nós em componentes do *cluster*. Com o *pacemaker* é possível disponibilizar quase que todas as configurações de redundâncias, dentre elas: *cluster* ativo/ativo, ativo/passivo, N-para-1 e N-para-N. (BEEKHOF, 2012).

As principais características do *pacemaker* são:

- Detecção e recuperação de falhas de nós e de nível de serviço;

- *Storage agnostic* (não é limitada a fabricante, podendo funcionar em vários tipos de sistema), sem necessidade de armazenamento compartilhado;
- *Resources agnostic* (não é limitada a fabricante, podendo funcionar em vários tipos de sistema), qualquer recurso pode ser configurado em *cluster*;
- Suporta *STONITH*¹⁵ para garantir a integridade dos dados;
- Suporta a grupos grandes e pequenos.

Segundo Beekhof (2012), o *pacemaker*, em conjunto com *corosync*, possibilita trabalhar com vários tipos de sistemas de arquivos clusterizados, além de definir se um nó está ativo (*up*) ou inativo (*down*).

2.5.4 CSYNC2

Segundo Wolf (2006), o *Csync2* é uma ferramenta para sincronização de arquivos em *clusters*. A sincronização de arquivos, de modo assíncrono, é indicado para os arquivos que sofrem poucas modificações, como arquivos de configuração ou imagens de aplicativos.

Essa ferramenta funciona de forma assíncrona, não precisando de estabelecimento de conexões persistentes entre os *hosts*. Quando os arquivos são atualizados em algum dos nós, eles são replicados automaticamente para os demais nós pertencentes ao *cluster*. Segundo Wolf (2006), Alguns profissionais de sincronização assíncrona são:

1. “Ferramentas de sincronização mais assíncronos são implementados como um único tiro comandos que precisam ser executadas de cada vez, a fim de executar um ciclo de sincronização. Por isso, é possível testar alterações em um host antes de implantá-los sobre os outros (e também retornar ao estado antigo, se as mudanças acabam por ser falso);”
2. “Os algoritmos de sincronização são muito mais simples e, portanto, menos propenso a erros;”

¹⁵*STONITH* é um mecanismo de segurança de *cluster*, que permite forçar o desligamento de nós que apresente problema. .

3. “Ferramentas de sincronização assíncrona pode ser (e geralmente são) implementadas como programas normais do modo de usuário. Ferramentas de sincronização síncrona precisam ser implementadas como extensões do sistema operacional. Portanto ferramentas assíncronas são mais fáceis de implantar e mais portátil;”
4. “É muito mais fácil para construir sistemas que permitem configurações com muitos hosts e regras de replicação complexas.”

As características do *Csync2* são de grande importância para ambientes em *cluster*, permitindo, como já mencionada, alterações na configuração dos serviços em apenas um nó, sendo os outros atualizados automaticamente pelo mecanismo de replicação, trazendo produtividade para os administradores de sistema.

3 IMPLEMENTAÇÃO DE SOLUÇÃO DE ALTA DISPONIBILIDADE PARA PROXY HTTP UTILIZANDO FERRAMENTAS LIVRE

Neste capítulo será demonstrado o desenvolvimento deste projeto, especificação do ambiente, instalação e configuração das ferramentas utilizadas para implementar uma solução de alta disponibilidade para proxy http utilizando ferramentas livre.

3.1 METODOLOGIA ADOTADA

Foi adotada uma metodologia prático-experimental, realizando a configuração de todas as ferramentas em ambiente de servidores virtualizados. Para tanto, foram utilizadas as seguintes ferramentas:

- VMware VShere na versão 4.1;
- Sistema operacional Debian, versão 6;
- Pacemaker, versão 1.0.9;
- Corosync, versão 1.2.1;
- Csync2, versão 1.34;
- Squid, versão 2.7.

3.2 TOPOLOGIA DA SOLUÇÃO

O ambiente utilizado foi composto por cinco servidores, sendo dois destes servidores com função de `SERVIDOR DIRETOR`, configurados com tolerância a falhas. Os demais servidores foram configurados com *Squid* com função de *Proxy-cache*, como demonstrado na Figura 7.

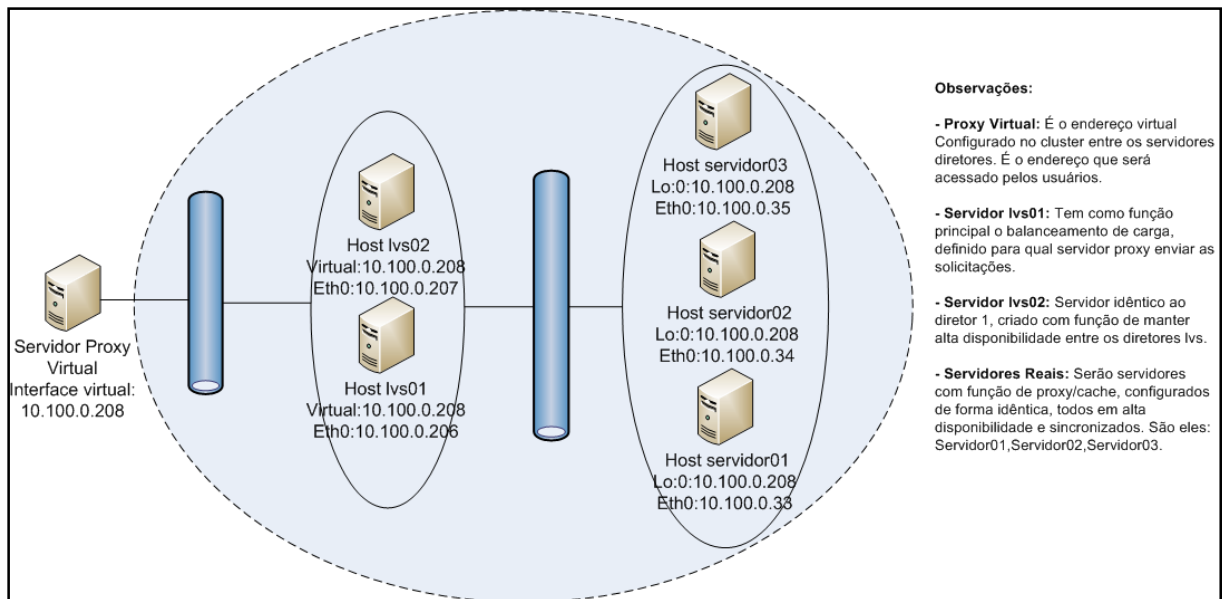


Figura 7 - Topologia completa da solução, com roteamento direto, contemplando balanceamento de carga e tolerância a falhas.

Como pode ser visto na Figura 7, a solução conta com cinco servidores, sendo **lvs01** e **lvs02**, idênticos, atuando como SERVIDORES DIRETORES, com função de balanceamento de carga e os **servidores 01, 02 e 03** funcionando como servidores de aplicação executando Squid-cache.

Como foi adotado o roteamento direto do LVS, todos os servidores, tanto os DIRETORES quanto os de aplicação possuem o endereço IP 10.100.0.208 em comum.

Outro aspecto importante é que os servidores diretores, além de redundantes, realizam a sincronização dos arquivos de configuração para os demais servidores, inclusive as configurações do squid, através das ferramentas corosync, pacemaker e csync2.

3.3 RESULTADOS

Nesta seção serão apresentados os testes que foram realizados, descrevendo as condições em que foram executados e os resultados obtidos. O ambiente utilizado para execução dos testes foi o mesmo apresentado na seção 3.2 onde foram simuladas e avaliadas as seguintes situações:

- Balanceamento de carga (regime normal de operação);

- Tolerância à falha em N-1 nó de aplicação (operação com falha em servidores de aplicação);
- Recuperação de falha em nó de aplicação;
- Tolerância à falha no balanceamento (operação com falha em SERVIDOR DIRETOR).

Para simular as requisições dos clientes foi utilizada ferramenta AB - *Apache Benchmark*, realizando 100 (cem) conexões simultâneas para o *cluster*.

3.3.1 Primeiro Experimento - Balanceamento de Carga

Neste experimento, SERVIDOR DIRETOR **lvs01** recebeu as conexões e realizou o balanceamento respeitando o algoritmo *Round Robin*, distribuindo as solicitações igualmente para os servidores de aplicação, como pode ser visto na Figura 8.

IP Virtual Server version 1.2.1 (size=4096)					
Prot	LocalAddress:Port	Scheduler	Flags		
->	RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
TCP	10.100.0.208:3128	rr			
->	servidor01:3128	Route	1	0	34
->	servidor02:3128	Route	1	0	33
->	servidor03:3128	Route	1	0	33

Figura 8 - Balanceamento de carga de 100 (cem) conexões, utilizando algoritmo *Round Robin*, para os servidores de aplicação do cluster.

Como é possível observar na Figura 8, todos servidores receberam o mesmo peso (*Weight*), ou seja, 1 (um). Essa condição indica que todos os servidores estão ativos e operacionais. Caso o peso (*Weight*) seja definido como 0 (zero), isso significa que o servidor está indisponível e não receberá requisições.

Para este experimento, mesmo com um volume baixo de requisições, podemos afirmar que a solução se comportou como esperado.

3.3.2 Segundo Experimento – Tolerância a Falha em (N-1) Nós de Aplicação

Neste experimento, foi removido o **servidor03**, e a solução funcionou de acordo com o esperado, ou seja, distribuiu as solicitações entre os servidores ativos, como pode ser visto na Figura 9.

IP Virtual Server version 1.2.1 (size=4096)					
Prot	LocalAddress:Port	Scheduler	Flags		
->	RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
TCP	10.100.0.208:3128	rr			
->	servidor01:3128	Route	1	0	50
->	servidor02:3128	Route	1	0	50
->	servidor03:3128	Route	0	0	0

Figura 9 - Balanceamento de carga de 100 (cem) conexões, com falha em um servidor de aplicação do *cluster*.

Como demonstrado na Figura 10, foi fixado o peso (*Weight*) 0 (zero) para o **servidor03**, indicando a sua indisponibilidade. Com isso, a carga de 100 conexões foi balanceada para os demais servidores (**servidor01 e servidor02**), mantendo o serviço disponível.

Para este experimento, novamente, podemos afirmar que a solução se comportou como esperado.

3.3.3 Terceiro Experimento – Recuperação de Falha em Nó de Aplicação

Após a remoção do **servidor03** no segundo experimento, foi simulada uma recuperação de falha recolocando o **servidor03** novamente em produção. Como é possível observar na Figura 10, **SERVIDOR DIRETOR lvs01** voltou a encaminhar normalmente solicitações para **servidor03**.

IP Virtual Server version 1.2.1 (size=4096)					
Prot	LocalAddress:Port	Scheduler	Flags		
->	RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
TCP	10.100.0.208:3128	rr			
->	servidor01:3128	Route	1	0	33
->	servidor02:3128	Route	1	0	34
->	servidor03:3128	Route	1	0	33

Figura 10 - Recuperação de falha em nó de aplicação.

Com todos os servidores funcionando normalmente, como pode ser visto na Figura 10, o balanceamento foi normalizado e todos os servidores passaram a receber requisições igualmente.

3.3.4 Quarto Experimento – Tolerância a Falha em Nó Balanceador de Carga

Neste experimento, o SERVIDOR DIRETOR **lvs01** foi desligado, simulando uma falha. A solução funcionou de acordo com o esperado, ou seja, foi realizado o chaveamento automático para o SERVIDOR DIRETOR **lvs02**, como pode ser visto na Figura 11.

```

=====
Last updated: Wed Sep 19 10:35:47 2012
Stack: openais
Current DC: lvs02 - partition WITHOUT quorum
Version: 1.0.9-74392a28b7f31d7ddc86689598bd23114f58978b
2 Nodes configured, 2 expected votes
1 Resources configured.
=====

Online: [ lvs02 ]
OFFLINE: [ lvs01 ]

FAILOVER-ADDR02 (ocf::heartbeat:IPaddr2):      Started lvs02

```

Figura 11 – Chaveamento automático do lvs01 para o lvs02, evitando que o *cluster* ficasse indisponível.

Como pode ser visto na Figura 11, o SERVIDOR DIRETOR **lvs02**, que estava definido como secundário, foi redefinido para primário, recebendo todas as solicitações e balanceando-as para os demais servidores de aplicação, evitando indisponibilidade do *cluster* de aplicação.

Além de tolerar uma falha, essa estrutura possui recurso automático a recuperação da falha, ou seja, no momento em que o servidor **lvs01** se tornar ativo novamente, será detectado como *online* e reassumirá sua condição de servidor primário.

Para este experimento, mais uma vez, podemos afirmar que a solução se comportou como esperado.

4 CONSIDERAÇÕES FINAIS

Neste trabalho foi realizado um estudo sobre alta disponibilidade em serviços, bem como quais ferramentas livres existem atualmente disponíveis para implementar um sistema de Proxy-cache para acesso à internet com alta disponibilidade, alto desempenho, confiável e de baixo custo.

Neste sentido, foram realizados estudos para conhecer quais ferramentas poderiam ser utilizadas e quais tipos de *clusters* poderiam ser implementados. Reunido o conhecimento e selecionadas as ferramentas, foi desenhada, implantada e testada uma solução de alta disponibilidade para serviço de Proxy utilizando ferramentas livres.

Assim, combinando conhecimento científico e ferramentas como *pacemaker*, *corosync* e LVS, demonstrou-se que é possível e viável a implementação de ambientes de alta disponibilidade que sejam versáteis, ou seja, que podem ser utilizado com serviço de Proxy HTTP, servidores web, correio eletrônico, etc. Vale ressaltar que, apesar de todas essas vantagens, a solução possui algumas limitações quanto as conexões persistentes, não as mantendo no momento de falha em algum dos nós da solução, assim tornando necessária uma nova conexão.

Sistemas de alta disponibilidade é uma realidade e serão cada vez mais demandados pelas áreas estratégicas das organizações, reforçando ainda mais a relevância desse tema e as contribuições apresentadas neste trabalho.

4.1 CONTRIBUIÇÕES DO TRABALHO

Este trabalho poderá servir como fonte de consulta teórica e prática, e seu tutorial poderá ser útil tanto para estudantes como para profissionais que desejam implementar um ambiente altamente disponível.

Destacam-se como principais contribuições desse trabalho, além da revisão bibliográfica realizada:

- Demonstrativo teórico e pratico de ambiente de alta disponibilidade;
- Tutorial, prático detalhado em português;

4.2 TRABALHOS FUTUROS

Vários desdobramentos e extensões deste trabalho podem ser realizados, tanto no aprimoramento de implementação de solução de alta disponibilidade para proxy http utilizando ferramentas livre, como também sua aplicação a outras áreas. Como melhorias possíveis citar:

- Implementação de solução de alta disponibilidade para servidores de aplicação Web;
- Desenvolvimento de Front-End para solução de cluster com software livre;

REFERÊNCIAS BIBLIOGRÁFICAS

- ANGUS, S. Linux man page. Disponível em: <<http://linux.die.net/man/8/corosync>>. Acesso em: 15 Junho 2012 as 17h35min.
- BEEKHOF, A. clusterlabs.org, 2012. Disponível em: <http://www.clusterlabs.org/doc/en-US/Pacemaker/1.1/pdf/Pacemaker_Explained/Pacemaker-1.1-Pacemaker_Explained-en-US.pdf>. Acesso em: 04 ago. 2012.
- DALHEIMER, K. et al. **Running Linux**. 4th. ed. [S.l.]: O'Reilly, 2002.
- GNU.ORG. **Projeto GNU**. Disponível em: <<http://www.gnu.org/philosophy/free-sw.html>>. Acesso em: 31 Agosto 2012.
- JOSEPH, M. LVS-mini-howto. **LVS-mini-howto**, 2004. Disponível em: <<http://www.austintek.com/LVS/LVS-HOWTO/mini-HOWTO/LVS-mini-HOWTO-pt.html>>. Acesso em: 12 ago. 2012.
- REDHAT. docs.redhat.com. **docs.redhat.com**. Disponível em: <http://docs.redhat.com/docs/pt-BR/Red_Hat_Enterprise_Linux/5/html/Virtual_Server_Administration/s1-lvs-scheduling-VSA.html>. Acesso em: 28 jul. 2012.
- SILVA, G. M. D. guiafoca.org. **Guia Foca Linux**, 2002. Disponível em: <http://www.guiafoca.org/cgs/guia/inic_interm/ch-kern.html>. Acesso em: 02 Setembro 2012.
- WESSELS, D. **Squid: The Definitive Guide**. [S.l.]: O'Reilly, 2004.
- WIKIPÉDIA. Failover. **Failover**. Disponível em: <<http://en.wikipedia.org/wiki/Failover>>. Acesso em: 28 jul. 2012.
- WIKIPÉDIA. Software Livre. **Software Livre**. Disponível em: <http://pt.wikipedia.org/wiki/Software_livre>. Acesso em: 31 Agosto 2012.
- WOLF, C. oss.linbit.com. **oss.linbit.com**, 2012. Disponível em: <<http://oss.linbit.com/csync2/paper.pdf>>. Acesso em: 12 jul. 2012.
- D. Wessels, K. Claffy. "RFC 2186: Internet Cache Protocol (ICP), version 2". Junho 2012.
- D. Wessels, K. Claffy. "RFC 2188: Application Internet Cache Protocol (ICP), version 2". Junho 2012.

ANEXO I - TUTORIAL DE INSTALAÇÃO E CONFIGURAÇÃO

Este tutorial visa apresentar a instalação e configuração do ambiente de alta disponibilidade, como apresentado neste trabalho. Será utilizado servidores com sistema operacional Linux Debian 6.

Instalação das Ferramentas

Aqui será demonstrado de forma detalhada a instalação dos pacotes necessários para configuração do ambiente.

Instalação de pacotes necessários nos servidores diretores.

Para configuração do Linux Virtual Server (LVS) em HA, é necessária a instalação dos pacotes corosync e pacemaker como abaixo:

01	apt-get install corosync
02	apt-get install pacemaker -y
03	apt-get install csync2 -y
04	apt-get install ldirectord -y
05	apt-get install ipvsadm -y

Tabela 1 Instalação de Pacotes Servidores Diretores

A instalação destes pacotes deve ser realizada nos dois servidores, visto a necessidade dos dois servidores idênticos.

Instalação de pacotes necessários nos servidores reais Proxy/Cache.

Para configuração do ambiente Proxy/Cache se faz necessário a instalação dos seguintes pacotes:

01	apt-get install csync2
02	apt-get install squid3

Tabela 2 Instalação de Pacotes Servidores Reais

A instalação destes pacotes deve ser realizada em todos os servidores reais com função de Proxy/Cache.

Configuração das ferramentas

Aqui será demonstrado de forma detalhada a configuração das ferramentas utilizadas para criação de ambiente de alta disponibilidade.

Configuração dos servidores diretores.

Os servidores diretores responsáveis pelo gerenciamento de cluster de balanceamento devem ser configurados em alta disponibilidade entre eles, sendo um o host principal e o segundo em backup para o servidor primário.

Servidor LVS	lvs01
Interface Eth0	10.100.0.206
Interface virtual	10.100.0.208

Tabela 3 Interfaces servidor lvs01

Servidor LVS	lvs02
Interface Eth0	10.100.0.207
Interface virtual	10.100.0.208

Tabela 4 Interfaces servidor lvs02

Configurando corosync

A configuração destes dois aplicativos não é necessário grandes alterações, a criação dos arquivos `corosync.conf` e o `pcmk`. O `corosync` deve ficar com a configuração abaixo:

01	totem {
02	version: 2
03	secauth: off
04	threads: 0
05	interface {
06	ringnumber: 0
07	bindnetaddr: 10.100.0.0
08	mcastaddr: 226.94.1.1
09	mcastport: 5405
10	}
11	}

Tabela 5 Arquivo Configuração corosync

Onde tem como alguma de suas principais variáveis destacadas como as seguintes: **secauth**, na linha 03, habilita criptografia, todas as mensagens serão autenticadas e criptografadas com HMAC-SHA1. Por padrão é habilitado on.

Threads, na linha 04, definem quantos seguimentos serão utilizados para criptografia e envio de mensagens. Por padrão o valor é zero.

Interface, na linha 05, é um subdiretório dentro do totem possuindo quatro parâmetros *ringnumber*, *bindnetaddr*, *mcastaddr*, *mcastport*.

Bindnetaddr, na linha 07, é especificado o endereço de rede que será usado.

Mcastaddr, na linha 08, é especificado o endereço multicast, é importante evitar endereço multicast, começando com 224.x.x.x.

Mcastport, na linha 09, é especificado o endereço de porta multicast.

O arquivo de configuração deve ser criado dentro do diretório da ferramenta /etc/corosync. É também necessário a criação do arquivo pcmk dentro dos diretórios de serviços do corosync /etc/corosync/services.d. segue informações do arquivo.

01	service {
02	name: pacemaker
03	ver: 0
04	}

Tabela 6 Configuração Corosync

Para inicialização do serviço corosync durante o boot do servidor é necessário a configuração do sistema no arquivo /etc/default/corosync, a variável deve estar como demonstrado logo abaixo.

01	START=yes
----	-----------

Tabela 7 Configurando Corosync para Boot.

É necessária a criação de uma chave de autenticação, para realizar comunicação entre os servidores pertencentes ao cluster. Esta chave deve ser compartilhada entre os dois servidores do cluster de alta disponibilidade.

01	# corosync-keygen
----	-------------------

Tabela 8 Gerando chave e Corosync

Duas outras funções que devem ser alterada são o stonith e quorum. Como será configurado apenas dois nós não se faz necessário o uso, visto que o quorum seria interessante em casos de cluster com mais de dois nós.

01	crm configure property stonith-enabled="false" no-quorum-policy="ignore"
----	--

Tabela 9 Configurando stonith e quorum.

O quorum tem a função em cluster de alta disponibilidade de habilitar as máquinas pertencentes ao cluster são quem decidirá qual servidor irá entrar em produção caso o servidor primário se torne inacessível.

Realizado as devidas configurações é possível verificar se a checagem entre os hosts esta sendo realizada com o comando abaixo:

01	# crm_mon
----	-----------

Tabela 10 Verificando monitoramento.

Com o comando acima é gerada uma interface de monitoramento do cluster, como demonstrado na tabela abaixo.

01	=====
02	Last updated: Wed Jul 4 13:19:17 2012
03	Stack openais
04	Current DC: lvs01 - partition with quórum
05	Version: 1.0.9-74392a28b7f31d7ddc86689598bd23114f58978b
06	2 Nodes configured, 2 expected votes
07	0 Resources configured.
08	=====
09	Online: [lvs01 lvs02]

Tabela 11 Tela de monitoramento Cluster.

Configurando pacemaker

Como demonstrado anteriormente, na tabela 11, é possível observar que os hosts em comunicação e monitorados. Com o pacemaker será possível o gerenciamento do cluster. Para criação do cluster é necessária a criação do ip virtual, no qual será utilizado, como demonstrado abaixo:.

01	# crm configure primitive ip-virtual01 ocf:heartbeat:IPaddr2 paramsip="10.100.0.208" op monitor interval="10s" meta is-managed="true"
----	---

Tabela 12 Configurando Cluster com pacemaker.

Como o foco principal destes servidores LVS, com objetivo de balanceamento de carga, para os servidores reais configurados com funções de Proxy. É importante a adição do serviço LVS, monitorando o arquivo de configuração /etc/ldirectord.cf. como abaixo:

01	crm configure primitive ldirectord ocf:heartbeat:ldirectord paramsconfigfile="/etc/ldirectord.cf" op monitor interval="5s" timeout="20s" meta target-role="Started"
----	---

Tabela 13 Configurando Cluster , checagem ldirectord.conf.

Realizada estas configurações os dois hosts estão configurando em cluster de alta disponibilidade, podendo ser monitorados como já demonstrado.

	=====
	Last updated: Wed Jul 4 13:19:17 2012
	Stack: openais
	Current DC: lvs01 - partition with quórum
	Version: 1.0.9-74392a28b7f31d7ddc86689598bd23114f58978b
	2 Nodes configured, 2 expected votes
	0 Resources configured.
	=====
	Online: [lvs01 lvs02]
	Ip-virtual01 (ocf::heartbeat:IPaddr2): Started lvs01

Tabela 14 Tela de monitoramento com cluster configurado.

Configurando Linux Virtual Server

O LVS é responsável pelo balanceamento de carga, e sua configuração se resume em um arquivo de configuração o /etc/ha.d/ldirectord.cf a configuração para este processo se resume a configuração abaixo:

01	checktimeout=10
02	checkinterval=2
03	autoreload=no
04	logfile="/local0"
05	quiescent=yes
06	virtual=10.100.0.208:3128
07	real=servidor01:3128 gate
08	real=servidor02:3128 gate
09	real=servidor03:3128 gate
10	fallback=127.0.0.1:80 gate
11	service=squid
12	scheduler=rr
13	protocol=tcp
14	checktype=connect

Tabela 15 Arquivo de configuração ldirectord.conf.

Nesta configuração algumas variáveis são importantes ser destacada, como demonstrada a lista abaixo:

Checktimeout, na linha 01, responsável por verificar quando um servidor estiver inacessível.

checkinterval, na linha 02, responsável pela quantidade de tempo para intervalo de checagem em segundos.

autoreload, na linha 03, caso seja detectada alterações nos arquivos de configurações realizar reload da configuração.

logfile, na linha 04, diretório onde será salvo os logs do serviço.

Scheduler, na linha 12, responsável pelo tipo de balanceamento podendo ele ser Round Robin (rr),

Weighted Round Robin (wrr), Least-Connection (lc), Weighted Least-Connection (wlc)

Checktype, na linha 14, é o tipo de conexão que será utilizado para checagem do serviço podendo ser negotiate, connect, N, ping, off,on.

Configurando csync2

A criação do diretório de backup é importante, pois em alguns casos as alterações em arquivos podem causar erros, ou seja, necessário *rollback* da configuração, com a função de backup é possível retornar a configuração anterior.

01	# mkdir /backup/csync2
----	------------------------

Tabela 16 Criando diretório de Backup.

E criação de um certificado SSL, como demonstrado a seguir:

01	# openssl genrsa -out /etc/csync2_ssl_key.pem 1024
02	# openssl req -batch -new -key /etc/csync2_ssl_key.pem -out /etc/csync2_ssl_cert.csr
03	# openssl x509 -req -days 6000 -in /etc/csync2_ssl_cert.csr -signkey /etc/csync2_ssl_key.pem -out /etc/csync2_ssl_cert.pem

Tabela 17 Criando Certificado.

O csync2 exige que todos os hosts participantes do cluster possuam uma chave compartilhada, sendo ela criada de forma bastante simples como mostra o comando a seguir.

01	csync2 -k /etc/csync2.key
----	---------------------------

Tabela 18 Adicionndo certificado.

O csync2 necessita de um repositório para replicar as modificações, então como o projeto é direcionado aos servidores com squid3, é importante a criação de diretórios e arquivos de configuração do squid3.

01	# mkdir /etc/squid3
02	# touch /etc/squid3/squid.conf

Tabela 19 Criando Repositórios.

Todas as configurações alteradas neste arquivo de configuração são replicadas a todos os outros servidores pertencentes ao cluster.

O arquivo de configuração principal do csync2 é onde se configura os grupos de sincronismo, na tabela abaixo são criados três grupos sendo eles: proxyconf-lvs01, proxyconf-lv02 e o directorconfig. Onde proxyconf-lvs01e proxyconf-lv02 são responsáveis pela replicação dos servidores LVS para os servidores reais. O grupo directorconfig irá realizar a replicação entre os servidores LVS. Segue o arquivo logo abaixo.

01	group proxyconf-lvs01
02	{
03	host lvs01 (servidor01) (servidor02) (servidor03);
04	key /etc/csync2.key;
05	include /etc/squid3/squid.conf;
06	include /etc/csync2.cfg;
07	action
08	{
09	pattern /etc/squid3/squid.conf;
10	exec "/etc/init.d/squid3 reload";
11	logfile "/var/log/csync2_action.log";
12	}
13	backup-directory /backups/csync2;
14	backup-generations 5;
15	auto left;
16	}
17	group proxyconf-lv02
18	{
19	host lvs02 (servidor01) (servidor02) (servidor03);
20	key /etc/csync2.key;
21	include /etc/squid3/squid.conf;
22	include /etc/csync2.cfg;
23	action
24	{
25	pattern /etc/squid3/squid.conf;
26	exec "/etc/init.d/squid3 reload";
27	logfile "/var/log/csync2_action.log";
28	}
29	backup-directory /backups/csync2;
30	backup-generations 5;
31	auto left;
32	}
33	group directorconfig
34	{
35	host lvs-director1 lvs-director2;
36	key /etc/csync2.key;
37	include /etc/squid3/squid.conf;
38	include /etc/csync2.cfg;
39	include /etc/lldirectord.cf;
40	backup-directory /backup/csync2;
41	backup-generations 5;
42	auto first;
43	}

Tabela 20 Arquivo de configuração csync2.

Configuração dos servidores Proxy/Cache.

Os servidores reais, pertencentes ao cluster de balanceamento, servidores no qual a aplicação irá esta em produção. Neste caso é de função de Proxy/Cache. Segue a configuração dos servidores.

Servidor Real	Servidor01
Interface Eth0	10.100.3.33
Interface Lo:0	10.100.0.208

Tabela 21 Interfaces servidor real 01.

Servidor Real	Servidor02
Interface Eth0	10.100.3.34
Interface Lo:0	10.100.0.208

Tabela 22 Interfaces servidor real 02.

Servidor Real	Servidor03
Interface Eth0	10.100.3.35
Interface Lo:0	10.100.0.208

Tabela 23 Interfaces servidor real 03.

Configurando Squid3

Foi realizada configuração padrão do SQUID, seguindo a configuração abaixo.

01	cache_mem 128 MB
02	maximum_object_size 512 MB
03	minimum_object_size 0 KB
04	cache_swap_low 90
05	cache_swap_high 95
06	cache_dir ufs /var/spool/squid3 2048 16 256
07	cache_access_log /usr/local/squid/var/logs/cache.log
08	http_port 3128
09	#Autenticação
10	auth_param basic children 5
11	auth_param basic realm AUTENTICAÇÃO
12	auth_param basic credentialsttl 2 hours
13	auth_param basic casesensitive off

14	acl autenticados proxy_auth REQUIRED
15	acl manager proto cache_object
16	acl localhost src 127.0.0.1/255.255.255.255
17	acl to_localhost dst 127.0.0.0/8
18	acl SSL_ports port 873 # rsync
19	acl Safe_ports port 443 # https
20	acl Safe_ports port 80 # http
21	acl Safe_ports port 21 # ftp
22	acl Safe_ports port 443 563 # https, snews
23	acl Safe_ports port 70 # gopher
24	acl Safe_ports port 210 # wais
25	acl Safe_ports port 1025-65535 # unregistered ports
26	acl Safe_ports port 280 # http-mgmt
27	acl Safe_ports port 488 # gss-http
28	acl Safe_ports port 591 # filemaker
29	acl Safe_ports port 777 # multiling http
30	acl Safe_ports port 631 # cups
31	acl Safe_ports port 873 # rsync
32	acl Safe_ports port 901 # SWAT
33	acl purge method PURGE
34	
35	acl sitesbloqueados url_regex -i "/etc/squid/db/bloqueados"
36	acl palavrasproibidas url_regex -i "/etc/squid/db/proibidos"
37	acl sitespermitidos url_regex -i "/etc/squid/db/permitidos"
38	
39	http_access allow sitespermitidos
40	http_access allow localhost
41	refresh_pattern ^ftp: 1440 20% 10080
42	refresh_pattern ^gopher: 1440 0% 1440
43	refresh_pattern . 0 20% 4320
44	
45	http_access deny all

46	http_access allow manager localhost
47	http_access deny manager
48	http_access allow purge localhost
49	http_access deny purge
50	http_access allow localhost
51	http_reply_access allow all
52	icmp_access allow all

Tabela 24 Arquivo de configuração do Squid.

Feita a configuração é necessário a configuração dos diretórios citados nas ACLs acima, assim é necessário a criação do diretório e arquivos.

01	# mkdir /etc/squid/db
02	# touch /etc/squid/db/proibidos
03	# touch /etc/squid/db/bloqueados
04	# touch /etc/squid/db/permitidos

Tabela 25 Criando Arquivos de ACLS.

Configurando csync2

A configuração do csync2 nos servidores reais é basicamente A criação do diretório de backup é importante, pois em alguns casos as alterações em arquivos podem causar erros, ou seja, necessário *rollback* da configuração, com a função de backup é possível retornar a configuração anterior.

01	mkdir /backup/csync2
----	----------------------

Tabela 26 Criando diretório de backup.

E criação de um certificado SSL, como demonstrado a seguir:

01	#openssl genrsa -out /etc/csync2_ssl_key.pem 1024
02	#openssl req -batch -new -key /etc/csync2_ssl_key.pem -out /etc/csync2_ssl_cert.csr
03	#openssl x509 -req -days 6000 -in /etc/csync2_ssl_cert.csr -signkey /etc/csync2_ssl_key.pem -out /etc/csync2_ssl_cert.pem

Tabela 27 Criando chave csync2.