Tolerância a Falha Utilizando Ferramentas Livres

Marcos Paulo Rodrigues Nogueira da Silva[1], Marcelo Henrique Euzebio Batista1[1]

[1]FACENSA - Faculdade Cenecista Nsa. Senhora dos Anjos , Av José Loureiro da Silva, 1991 - 94.010-010 - Gravataí / RS

mprnds@terra.com.br, marcelo.batista@sinprors.org.br

Resumo. *Este artigo apresenta a utilização de ferramentas livres para empregar uma solução de tolerância a falhas na modalidade de failover, obtendo resultados totalmente satisfatórios com relação a custo e beneficio. Proporcionando comparações entre ferramentas que necessitam de licenciamento.*

1. Introdução

Em razão do alto custo de implementação do sistema de tolerância a falhas, foi desenvolvida uma solução que possibilita a minimização do valor alocado, utilizando como base o protocolo *CARP.*

Foram utilizadas as seguintes ferramentas: U*carp*; implementação do protocolo *CARP;* Ubuntu 8.1 (sistema operacional utilizado para o estudo de caso), com custo zero, softwares livres de código aberto.

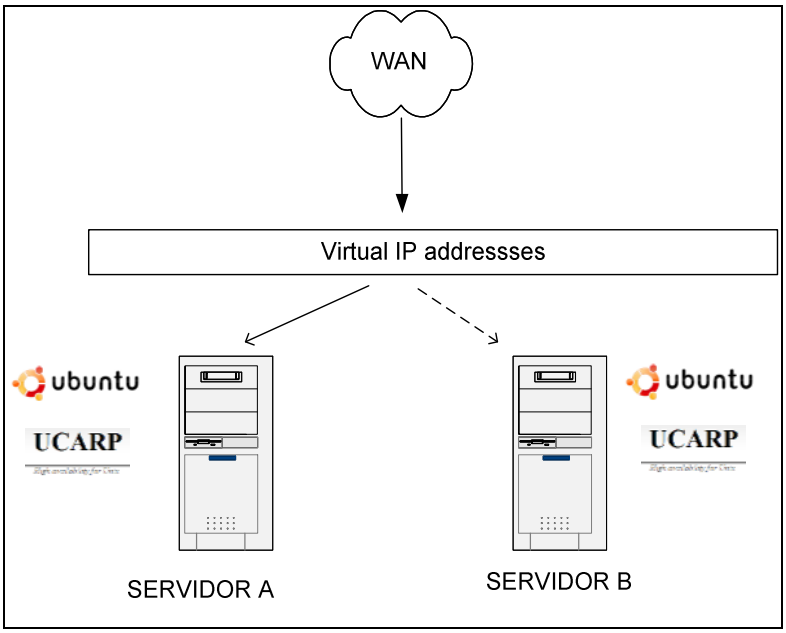

Figura 1. Estrutura proposta em uma solução de *failover*

Na estrutura demonstrada na Figura 1 teremos a possibilidade de utilizar diversos componentes, mantendo a tolerância em caso de falha de um dos nós.

2. Sistema Operacional

O sistema Ubuntu tem evoluindo através dos anos, tendo em cada nova versão melhoras significativas, relacionadas ao sistema base como seu *kernel* ou com itens relacionados a protocolos de segurança.

Para efetuar a instalação e uso desse sistema, propomos nesse trabalho a utilização de duas máquinas dedicadas para os testes. O download do sistema operacional referido pode ser efetuado através do link do distribuidor [Ubuntu, 2008].

Foi efetuado o download do arquivo através do link do distribuidor [Ubuntu, 2008] (.iso), havendo sido passado para um CD.

Inicia-se no computador ou máquinas virtuais a opção de *boot* através do drive de CD, efetuando a instalação do produto com as opções padrão.

A instalação foi executada em dois computadores, utilizando a seguinte configuração:

Tabela 1. Parâmetros de Hardware

Processador	Memória	Disco Local	Sistema Operacional
Pentium Dual Core , 1.46 GHz	1 G	120G	Ubuntu 8.1
Pentium Dual Core , 1.46 GHz	521 MB	40G	Ubuntu 8.1

3. Cluster em modalidade de *Failover*

Cluster é o nome dado para uma camada de disponibilidade onde poderá ser utilizado mais de um servidor, criando um ambiente de tolerância em caso de falhas.

Neste artigo aborda-se a modalidade de *failover*, identificando a falha do servidor principal e alterando a rota de acesso, conforme figura 2. No momento da falha o nó *slave* tomará o lugar do servidor *master*, sendo essa atividade transparente para o cliente.

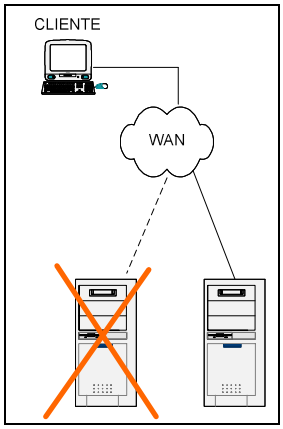

Figura 2. Estrutura de cluster na modelo de *failover*

Para podermos concluir a utilização de alta disponibilidade com cluster de *failover* com ferramentas *opensource,* iremos utilizar a tecnologia de protocolo *CARP* (*Common Address Redundancy Protocol*), que na tradução literal será Protocolo Comum de Endereço de Redundância. O *CARP* controla o *failover* na intersecção das

camadas 2 e 3 do modelo OSI , sendo elas a camada de ligação e de IP. [Tanembaum, 2003].

Na utilização desse protocolo teremos o programa chamado U*carp* que gerencia as questões de *failover* através de endereços MAC e agentes de monitoramento chamados *pfsync*.

Esse software é disponível na modalidade de *opensource*, sendo possível a utilização e modificação de seu código. O Software é escrito na linguagem C, sendo possível o entendimento de sua lógica e alteração se necessário para a sua utilização.

Segue um trecho de código do u*carp*, figura 3, onde é demonstrada sua programação.

```
static void carp_set_state(struct carp_softc *sc, int state)
{
    if ((int) sc->sc_state == state) {
        return;
    }
    switch (state) {
    case INIT:
        logfile(LOG_INFO, _("Switching to state: INIT"));
        break;
    case BACKUP:
        logfile(LOG_WARNING, _("Switching to state: BACKUP"));
  if ((sc->sc_state != INIT) || (neutral != 1)) {
      (void) spawn_handler(dev_desc_fd, downscript);
  }
        gratuitous_arp(dev_desc_fd, 0);
        break;
    case MASTER:
        logfile(LOG_WARNING, _("Switching to state: MASTER"));
        (void) spawn_handler(dev_desc_fd, upscript);
        gratuitous_arp(dev_desc_fd, 1);
        break;
    default:
        logfile(LOG_ERR, _("Unknown state: [%d]"), (int) state);
        abort();
    }
    sc->sc_state = state;
}
```

Figura 3. Exemplo de código utilizando u*carp*

O U*carp* considera em seus ambientes grupos de utilizadores, que serão identificados por IDs (identificadores), para ter mais de um IP virtual disponível no mesmo servidor. Cada grupo terá um endereço virtual, chamado de VIP (virtual IP), este IP será compartilhado sempre que necessário para manter a disponibilidade. Todo o endereço terá um MAC associado, que será a camada de ligação entre o virtual e o real.

Para a identificação de qual endereço o VIP deve responder, os anfitriões de cada grupo irão sinalizar pedidos ARP para o endereço virtual, atendendo requisições pelo MAC virtual. Todas as informações emitidas do ambiente interno para o externo irão levar o endereço da fonte.

Na sua instalação foi efetuada a seguinte seqüência: (1) conectado com o usuário root, em ambos servidores; (2) executado o comando apt-get install U*carp*, sendo necessário que os servidores estejam conectados à internet, para que seja possível o download dos arquivos imprescindíveis para instalação. Outrossim é possível obter os arquivos necessários no site da distribuição [U*carp* ,2008] .

O pfsync é o responsável por efetivar a inserção, *update* e limpeza na tabela de estado (informações físicas) entre as camadas do *firewall*. Cada *firewall* emite uma

mensagem para todos, através da propagação de protocolo em uma relação especifica, usando o protocolo de pfsync (protocolo 240 do IP).

Esse processo garante que o pfsync se encontre com as exigências do volume e da latência do pacote, a execução inicial não tem nenhuma autenticação interna.

Uma requisição efetuada pelo acesso local da camada de ligação ou pela subnet usada para o tráfego do pfsync, faz com que seja possível adicionar, mudar, ou remover os estados do *firewall*, sendo possível ativar o protocolo do pfsync em uma das redes "reais", mas devido aos riscos da segurança, recomenda-se uma rede dedicada (DMZ) e confiável que seja apenas usada para o pfsync.

4. Seqüência de Failover

Analisando uma linha do tempo dos eventos de um *failover* típico, considera-se a seguinte seqüência: inicialmente o nodo *master* propaga a informação para o meio referente à utilização do VIP para este endereço, após ser criado o *status* e atualizado, todas as requisições são encaminhadas para o nodo *master*.

Se em algum momento o servidor *master* falha, o servidor *slave* assume o papel. Quando a propagação do *carp* detecta que o meio não responde, o mesmo cria um novo status e atualiza o meio através de propagação e atualização na tabela do ambiente. Este tempo de atualização pode demorar em média até 3s.

Após o servidor *master* retornar ao seu estado normal, é disparada uma solicitação para todo o ambiente, requerendo a atualização das tabelas, para que sejam encaminhados os requerimentos diretamente para o nodo *master*. Lembrando que esta atualização de *failback* só irá ocorrer após toda a etapa de transação de atualizações ser concluída, sendo assim não gera nenhum tempo de indisponibilidade.

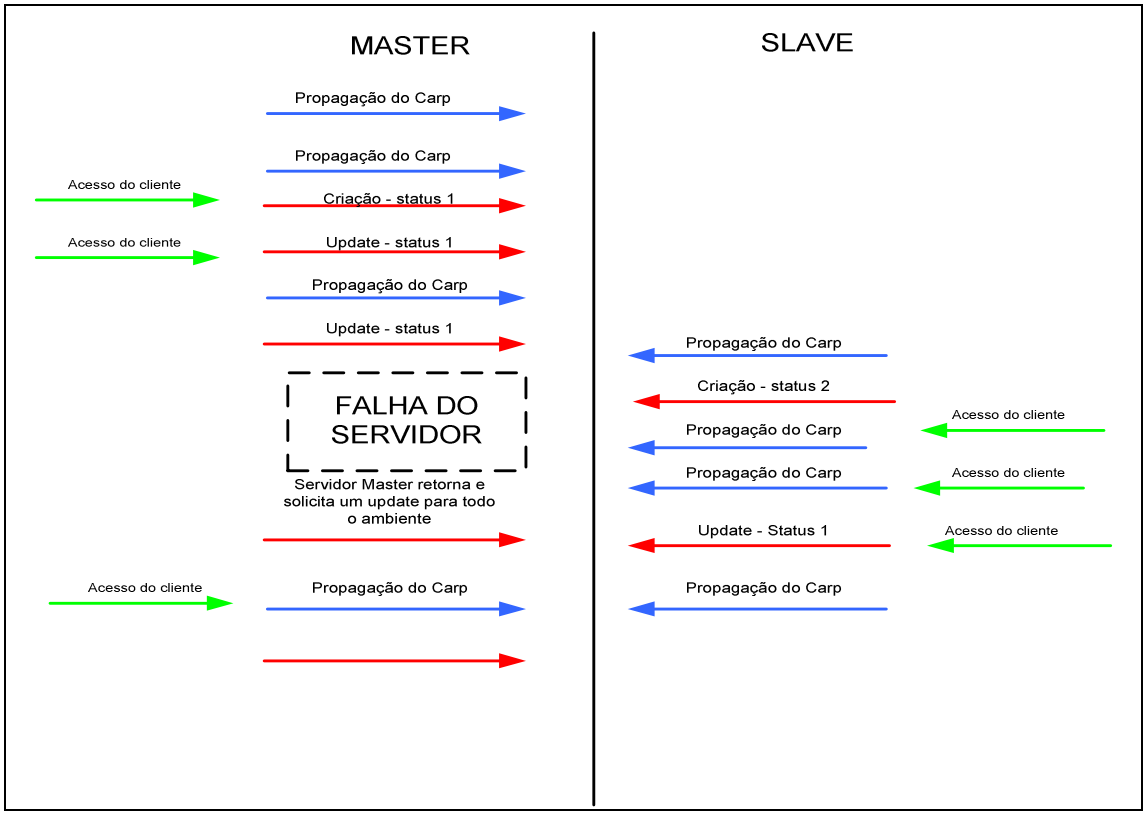

Figura 3. Linha do tempo de *failover*

5. Experimento

Efetivada a instalação dos sistemas operacionais e do U*carp*, foram executados os seguintes procedimentos, levando em consideração as seguintes informações, conforme tabela 2: (1) endereçamento do servidor *master* e *slave*; (2) configuração dos seguintes arquivos: (a) /etc/u*carp*/scripts/vip-up.sh; (b) /etc/u*carp*/scripts/vip-up.sh.

Tabela 2. IPs dos servidores

VIP (virtual IP)	192.168.0.10
Servidor Máster	192.168.0.50
Servidor Slave	192.168.0.51

Após a configuração dos arquivos, devem ser ativados os seguintes programas, conforme figuras 3, 4, 5, 6 e 7, pois demonstram a seqüência de ativação dos serviços. Deve-se colocar o arquivo em *background* utilizando o comando &, conforme o padrão de configuração do software U*carp*:

```
#! /bin/sh
/sbin/ip addr add 192.168.0.10/24 dev eth0

# Scripts de Startup se necessario
/etc/meu_sistema/start.sh
```

Figura 3. Script de startup do ambiente de cluster

```
#! /bin/sh
/sbin/ip addr del 192.168.0.10/24 dev eth0

# Scripts de Startup se necessario
/etc/meu_sistema/stop.sh
```

Figura 4. Script de stop do ambiente de cluster

```
ucarp -i <interface> -k 1 -s <vip> -v <id> -p <passowrd> -a <ip_local> -u <script_startup> -d <script_down> -B &
```

Figura 5. Modelo de chamada do software U*carp*

```
ucarp –i eth0 –P –k 1 –s 192.168.0.10 –v 28 –p manager –a 192.168.0.50 –u /etc/ucarp/vip-up.sh -d /etc/ucarp/vip-down.sh –B &
```

Figura 6. Ativação do nó master

```
ucarp –i eth0 –k 1 –s 192.168.0.10 –v 28 –p manager –a 192.168.0.51 –u /etc/ucarp/vip-up.sh -d /etc/ucarp/vip-down.sh –B &
```

Figura 7. Ativação do nó slave

Após estes comandos executados nos servidores correspondentes o ambiente já estará trabalhando em cluster. Com o ambiente concluído e em execução foi efetuado o seguinte teste, conforme resultado apresentado na tabela 3, o tempo médio de troca entre os servidores é de 3 segundos.

Nas tabelas 3 e 4 pode-se observar que o único ponto que podemos ter indisponibilidade é na falha do servidor *master*, sendo esse tempo muito curto e aceitável.

Tabela 3. Falha no servidor Master

Servidor	Falha	Ativo	Tempo	Ocorrência
Master	X			Desligamento forçado
Slave		X	3s	3s segundos para assumir o ambiente

Tabela 4. Retorno do servidor Master

Servidor	Falha	Ativo	Tempo	Ocorrência
Master		X	0s	Assume o ambiente sem interromper o acesso
Slave		X	0s	Retorna ao papel de Slave

6. Trabalhos relacionados

Entre os principais trabalhos pesquisados relacionados a soluções proprietárias, destacam-se: (1) Veritas Cluster Server, ferramenta de gerenciamento central, *failover* automatizado [Symantec, 2008]; (2) Red Hat Cluster Suite, ferramenta para serviços de *failover* [Red Hat, 2008].

O principal diferencial entre os trabalhos relacionados e o artigo apresentado é a questão do custo zero sobre licenciamento, pois basea-se em tecnologias de licença livre.

7. Conclusão

O principal objetivo desse artigo é demonstrar uma solução de cluster utilizando ferramentas de custo zero, como demonstra o aplicativo U*carp* que controla o ambiente, sendo de código aberto e gerando possibilidade para customização no que tange programação C.

As principais conclusões alcançadas mostram a grande possibilidade de implementações de alto nível e de grande produtividade com fácil acesso e controle.

O trabalho futuro terá uma estrutura utilizando banco de dados relacional e um serviço de aplicação com a utilização de *failover* existindo a possibilidade de criar persistências de banco de dados com custo zero, por exemplo, Oracle Database 10g Expression, tendo um banco de dados em cada nó do cluster e vinculando eles através da utilização de *dblinks*, sendo assim criando uma contingência tanto a suporte de servidor como a suporte de banco de dados.

8. Referências

Ryan McBride (2008), “Firewall Failover with pfsync and CARP”, http://www.countersiege.com/doc/pfsync-carp/

Red Hat Network (2008),” Low-cost high availability for enterprise applications”, http://www.redhat.com/cluster_suite/.

Symantec (2008),”Operações em infra-estrutura”, Symantec http://www.symantec.com/pt/br/business/solutions/key_products.jsp?solid=io

Tanembaum, Andrew S. Redes de Computadores 4ª edição 2003. Editora Campus

Torres, Gabriel Redes de Computadores Curso Completo editora Axcel Books, 2001

Ubuntu (2008),”System Operation”, http://www.ubuntu.com

Denis, Frank, “High availability for Unix”,http://www.ucarp.org